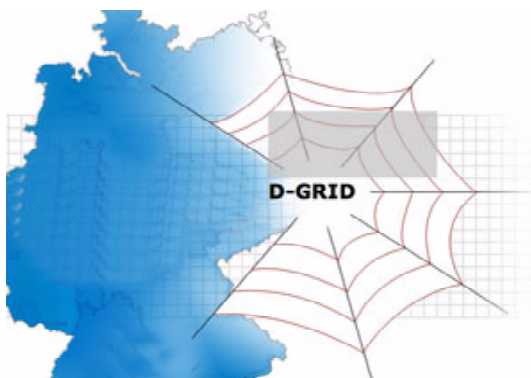




Universidad Nacional del Nordeste
Facultad de Ciencias Exactas, Naturales y Agrimensura

Monografía de Adscripción

Grid Computing



Aguilar Gladys Carolina- L.U.: 33.902

Prof. Coordinador: Agr. Castor Herrmann
Prof. Orientador: Mgter. David Luis la Red Martínez

Licenciatura en Sistemas de Información
Corrientes - Argentina

2005

A mi familia

A mis Amigos

A mi Profesor

Prefacio

Esta monografía trata aspectos fundamentales relacionados con el estudio del software de *Grid Computing* el cual permite el desarrollo de aplicaciones originales orientadas a la realización de cálculos intensivos.

En los últimos tiempos ha habido importantes avances tecnológicos relacionados con la posibilidad del uso del *Grid computing* aplicada a la resolución de problemas que requieren almacenamiento masivo, y/o cálculo intensivo. Muchas de estas aplicaciones tienen que ver con el desarrollo de modelos computacionales para la resolución de problemas de pronósticos meteorológicos, aplicaciones de la computación gráfica al diagnóstico por imágenes de alta resolución, el cálculo de números irracionales con millones de dígitos, la criptografía con números primos del orden de los 400 dígitos, etc.

Cuando la potencia de cómputo requerida no puede ser alcanzada localmente, es posible utilizar el grid para obtener potencia de cómputo de dispositivos que la puedan brindar remotamente.

Dicho esto, el objetivo definido en este contexto es la realización de una amplia investigación, que tuvo a su vez como objeto principal el desarrollo de una aplicación utilitaria para asistir a los usuarios en el análisis automático e inteligente de la información brindando de esta manera nuevas herramientas para el procesamiento de aplicaciones de cálculo intensivo.

Aguilar G. Carolina
Licenciatura en Sistemas de Información
Universidad Nacional del Nordeste
Corrientes; 15 de septiembre 2005

Índice General

1	Introducción	1
1.1	Concepto de Grid Computing	1
1.1.1	Qué Es el Grid	1
1.1.2	Beneficios Que Ofrece el Grid Computing	3
1.2	Introducción a Grid Computing	4
1.3	¿Qué Es y Para Qué Sirve el Grid Computing?	5
1.4	Arquitectura del Grid	9
1.5	Aplicaciones y Servicios en El Grid	12
1.5.1	Supercomputación	12
1.5.2	Proceso Intensivo de Datos	13
1.6	¿Qué es el Globus?	14
1.7	Estándares Abiertos	15
1.7.1	Web Service: Servicios Web	16
1.7.2	Grid Service: Servicios Grid	16
1.7.3	Open Grid Services Architecture: Arquitectura de Ser- vicios de Grid Estándar (OGSA)	17
1.7.4	Open Grid Services Infrastructure: Infraestructura de Servicios de Grid Estándar (OGSI)	18
1.7.5	Las tendencias futuras de los Servicios Grid	18
1.8	El Significado de Habilitar un Grid	20
1.8.1	Estratégicamente los Servicios Web Llegan Primero	20
1.9	Introducción al IBM Grid Toolbox	21
1.9.1	Los Objetivos del IBM Grid Toolbox	21
1.9.2	Soporte y Disponibilidad de Plataforma	22
1.9.3	Una Visión General de los Componentes del IBM Grid Toolbox	22
1.9.4	El Ambiente Hosting	24
2	Principios de Grid Computing	27

2.1	Lo Que el <i>Grid Computing</i> Puede Hacer	27
2.1.1	Aprovechamiento de los Recursos Que No Siempre Se Usan	27
2.1.2	La Capacidad de CPU Paralela	29
2.1.3	Las Aplicaciones	30
2.1.4	El Acceso a los Recursos Adicionales	31
2.1.5	Balanceo de Recursos	31
2.1.6	Confiabilidad	32
2.1.7	Administración	34
2.2	Los Conceptos y Componentes del Grid	34
2.2.1	Los Tipos de Recursos	34
2.2.2	Computación	35
2.2.3	Almacenamiento	36
2.2.4	Las Comunicaciones	37
2.2.5	El Software y las Licencias	39
2.2.6	El Equipo Especial, Capacidades, Arquitecturas, y Políticas	39
2.2.7	Los Trabajos y las Aplicaciones	40
2.2.8	Scheduling, Reservación, y Barrido	41
2.3	Intragrid a Intergrid	42
2.4	Construcción del Grid	45
2.5	Planificación del Despliegue	45
2.5.1	Seguridad	45
2.5.2	Organización	46
2.6	Componentes del Software Grid	46
2.6.1	Componentes de Administración:	47
2.6.2	Software Servidor	47
2.6.3	Software de Sumisión	49
2.6.4	Administración del Grid Distribuido	49
2.6.5	Schedulers	49
2.6.6	Las Comunicaciones	51
2.6.7	Observación, Dirección, y Decisión	51
2.7	Usar un Grid: Perspectivas de Usuario	52
2.7.1	Conectar e Instalar el Software de Grid	52
2.7.2	Registrarse En el Grid	53
2.7.3	Solicitar y Realizar Trabajos.	54
2.7.4	Configuración de Datos	56
2.7.5	Monitoreo del Progreso y Recuperación	57
2.7.6	Reservar Recursos	58

2.8	Usar un Grid: La Perspectiva de Un Administrador	58
2.8.1	Planeación	58
2.8.2	Instalación	59
2.8.3	Matriculación de Dirección	60
2.8.4	Certificado de Autoridad	61
2.8.5	Administración de Recursos	63
2.8.6	Compartir Datos	63
2.9	Usar Un Grid:Una Perspectiva del Diseñador de la Aplicación	64
2.10	El Presente y el Futuro del Grid	65
2.10.1	¿Qué No Puede Hacer el <i>Grid Computing</i> ?	66
3	Definición del Caso Para el Grid	67
3.1	Todo Acerca de On Demand	67
3.1.1	¿Por Qué Prestar Atención a On Demand?	69
3.1.2	Ambiente Operativo	69
3.1.3	Todo Acerca de la Evolución	70
3.1.4	La Propuesta Clave	71
3.2	Los Grado de Adopción de Grid	72
3.2.1	Las Estrategias Para la Habilitación de Grid	72
3.2.2	Software de Infraestructura Grid Orientado a Batch	75
3.2.3	Software de Infraestructura Grid Basado en SOA	76
3.3	1º, 2º y 3º Grado de Adopción	77
3.3.1	Batch Anywhere: 1º Grado de Adopción.	77
3.3.2	Independent Concurrent Batch	84
3.3.3	Parallel Batch	86
3.4	Habilitación de Código Existente	92
3.4.1	Existe un Modelo	92
3.4.2	Los Escenarios Conocidos	93
3.5	4º, 5º y 6º Grado de Adopción	108
3.5.1	Servicios (4º Grado)	108
3.5.2	Servicios Paralelos (5º Grado)	115
3.5.3	Programas Paralelos Completamente Acoplados (6º Gra- do)	120
3.6	En Resumen	124

Índice de Figuras

1.1	Representación de un grid como un grafo.	2
1.2	Origen y desarrollo del Grid Computing.	6
1.3	Características de un Grid Computing.	7
1.4	Aplicaciones de la computación en Grid.	9
1.5	Arquitectura del grid.	11
1.6	Arquitectura actual de los sistemas de información.	13
1.7	Arquitectura “Grid Computing” para sistemas de información.	14
1.8	Convergencia de Servicios Web y Grid.	19
1.9	Macro componentes.	24
1.10	Ambiente hosting.	25
2.1	Balaceo de Recursos.	32
2.2	Cofiabilidad en los Sistemas de Grid.	33
2.3	Administración en los Sistemas de Grid.	35
2.4	Almacenamiento en el Grid.	38
2.5	Trabajos y Aplicaciones del Grid.	40
2.6	Un Grid más Simple.	44
2.7	Un Intergrid más Complejo.	44
3.1	Camino a la Visión On Demand.	68
3.2	Los Principios en On Demand.	69
3.3	De Administrativo a la Productividad Organizacional.	71
3.4	Estrategias de Habilitación de Grid para Arquitecturas Orientadas al Servicio y en Lotes (Batch).	72
3.5	1º Grado de Adopción.	77
3.6	2º Grado de Habilitación Grid.	84
3.7	3º Grado de adopción del Grid.	87
3.8	Modelo de Integración Para Habilitar Código Existente.	93
3.9	Despliegue de Aplicaciones Monolíticas Usando el Modelo de Habilitación Estándar.	94

3.10 El Despliegue Ideal Para las Aplicaciones Modulares.	97
3.11 El Escenario Más Común Para el Despliegue de Aplicaciones Modulares.	99
3.12 El Modelo Estructural Servlet-centric más Común.	100
3.13 Estrategia Típica de Habilitación Web Servlet-centric.	101
3.14 Despliegue grid típico de Servlet central.	102
3.15 Modelo arquitectónico más común Database-centric.	104
3.16 Escenario típico de habilitación web Database-centric.	105
3.17 Escenario de virtualización de datos para aplicaciones Database- centric.	107
3.18 4º Grado de adopción.	109
3.19 5º Grado de adopción de Grid.	116
3.20 6º Grado de adopción de Grid	122

Capítulo 1

Introducción

1.1 Concepto de Grid Computing

El concepto principal de *Grid Computing* es el de compartir potencia computacional [2, Berstis].

Desde el momento en el que los primeros ordenadores comenzaron a conectarse a Internet, surgió la idea de unir la potencia inutilizada de cada uno para abordar problemas a los que sólo podían enfrentarse las supercomputadoras pertenecientes a organizaciones gubernamentales, universidades o grandes multinacionales.

La tecnología que hace esto posible se llama *Grid Computing* (malla de ordenadores) y se basa en el aprovechamiento de los ciclos de procesamiento no utilizados por los millones de ordenadores conectados a la red, logrando de esta forma la realización de tareas que son demasiado intensivas para ser resueltas por una única máquina.

1.1.1 Qué Es el Grid

El grid permite la *Computación Distribuida* sobre de una red de recursos heterogéneos utilizando estándares abiertos.

En esencia, el grid es una estructura de recursos que se virtualizan para optimizar los propósitos. Esto significa mayor poder de CPU, aplicaciones,

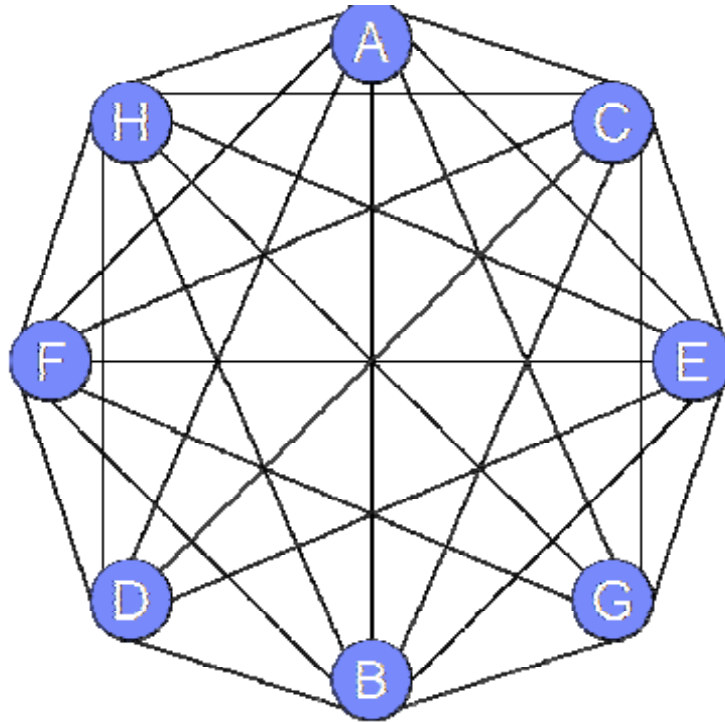


Figura 1.1: Representación de un grid como un grafo.

datos, almacenamiento y recursos de red. Está para que el cliente la utilice cuando sea necesario, sin preocuparse de dónde se localiza, en qué sistema operativo residen los datos y demás recursos, etc.

Teóricamente, según el *National Institute of Standards and Technology* : Instituto Nacional de Estándares y Tecnologías, el grid es “.....un grafo con una unión entre cada par de vértices”. Como se ilustra en la figura 1.1 de la página 2.

En los grafos cada nodo se conecta a los otros nodos de la siguiente manera : un enlace entre cualquier par de nodos. Esto significa que desde un nodo A a cualquier otro nodo implica el mismo esfuerzo, distancia, tiempo o cualquier otro factor representado por los enlaces del grafo. Indiscutiblemente, esta es la manera en que las tareas del grid trabajan utilizando el concepto de virtualización.

Los nodos y los enlaces pueden representar virtualmente cualquier cosa. En el caso de los grid de computación, el desafío consiste en acceder a los recursos que residen en cualquier parte de la estructura.

La estructura puede ser Intranet, Extranet o bien un ambiente de Internet. En otros términos, si se supone que el grafo ilustrado en la figura 1.1 de la página 2 es un *Grid Computacional*, el grafo representaría lo fácil que es, por ejemplo, para una máquina A obtener y utilizar los recursos de las máquinas B, C, D, o H. Los nodos son las máquinas y los enlaces representan la accesibilidad, disponibilidad y facilidad de uso. En términos simples, los enlaces representan la virtualización.

Construir un grid es un problema de software. Para una aplicación, ejecutarse en un grid o habilitarla en un grid ya hecho, significa ejecutarla en un *Ambiente Distribuido* que se comporta como un grafo completo.

1.1.2 Beneficios Que Ofrece el Grid Computing

Los principales beneficios son:

- Ofrecer flexibilidad para llenar las necesidades cambiantes del negocio.
- Brindar alta calidad a menor costo.
- Facilitar el pronto retorno de las inversiones.
- No necesitar de toda una nueva infraestructura para que funcione.
- Facilitar poder de computación / precio muy barato.
- Brindar el poder de un supercomputador.
- Utilizar software gratuito y usar código fuente abierto.
- No precisar hardware adicional, para posibilitar el incremento de la potencia de cómputo.
- Brindar transparencia para el usuario que participa en el grid.

1.2 Introducción a Grid Computing

Las supercomputadoras ciertamente tienen un papel que cumplir, pero son caras y tienden a ser muy especializadas en los tipos de problemas que se les presenta para resolver. El cambio de usar una única computadora para resolver el problema, a usar múltiples computadoras, es un cambio muy dramático de paradigma.

Profesionales en áreas de interés relativas a la computación de altos requerimientos tales como análisis sísmico, medicina, modelado financiero, investigación biomédica, y predicción del tiempo, se ven obligados a crear continuamente nuevas tecnologías para soportar sus cargas de trabajo. Desafortunadamente con frecuencia terminan por escoger soluciones que solo resuelven un problema específico dentro de un ambiente muy acotado de hardware o software.

La mayoría de los trabajos grid que se efectuaron apuntaron al ambiente de la informática distribuida de alto rendimiento, muy especializados, los cuales no podían utilizarse para aplicaciones informáticas distribuidas genéricas y heterogéneas. Como el grid se convierte en un factor muy importante para la informática comercial, las necesidades de estandarizaciones y de aplicaciones abiertas son cruciales.

Para subsanar los inconvenientes antes mencionados se formó el *Global Grid Forum: Foro Global de Grid (GGF)* para diseñar y desarrollar un conjunto de estándares de *Grid Computing* que podrían usarse para cualquier industria, en áreas geográficamente dispersas, hardware y ambientes de software heterogéneos.

Las estandarizaciones necesarias fueron identificadas en áreas tales como la arquitectura global, el modelo de programación, el modelo de red, el modelo de datos, la seguridad y la administración.

La idea de grid está enfocada fundamentalmente en el acceso remoto a recursos computacionales y pretende ser un paradigma de desarrollo “**no centrado en una tecnología concreta**”.

La evolución de *Grid Computing* se refleja en el avance de la estandarización de esta tecnología (el estándar de *Globus Project* es el estándar de facto) donde se encuentra definida la arquitectura del grid, los niveles de acceso, los requisitos, los servicios, etc.

El *Grid Computing* se enmarca dentro de la tecnología de computación distribuida englobando conceptos como sistemas operativos distribuidos, programación multiprocesador, redes de ordenadores, computación paralela, redes de computadoras, seguridad, bases de datos, etc. De cierta forma da una unidad conceptual a estos problemas de manera que todos ellos puedan verse desde una perspectiva grid.

La implementación de los estándares de grid ha permitido que diseñadores y administradores se centraran en problemas de un área específica, admitiendo así la utilización de una infraestructura grid cuando ellos lo requieran como cualquier otra herramienta portable. Esto reduce el costo del proyecto global y significativamente también el tiempo de despliegue de la aplicación.

El *Grid Computing* es más que una idea ambiciosa, ya que no sólo se trata de compartir ciclos de CPU para realizar cálculos complejos sino que se busca la creación de una infraestructura distribuida. Esta ardua tarea involucra labores de definición de la arquitectura general, de interconexión de diferentes redes, de definición de estándares, de desarrollo de procedimientos para la construcción de aplicaciones, etc.

El grid es una idea que promete revolucionar el mundo de la computación y el cómo se desarrollan las aplicaciones actualmente.

Parece claro que *Grid Computing* no es una moda, y queda, todavía, un largo camino por recorrer desde los conceptos hasta las aplicaciones reales. Aunque existen muchos mini - grids para el desarrollo de investigaciones no parece cercano el día en que todos los ordenadores del mundo formen un *Grid Mundial* a modo de gigantesco sistema de distribución eléctrico (paradigma del *Grid Computing* en casi todas las publicaciones sobre el tema), donde los usuarios se conecten y tengan acceso a la capacidad de cómputo y de almacenamiento que precisen sin preocuparse de dónde se genera. Ver figura 3.1 de la página 68.

1.3 ¿Qué Es y Para Qué Sirve el Grid Computing?

El concepto de *Grid Computing* da idea de una gran potencia de cálculo y almacenamiento, y parece un gran avance en las ciencias de la computación.

Existen además una multitud de aplicaciones reales que hacen uso de mini - grids, casi todas ellas están centradas en el campo de la investigación en el



Figura 1.2: Origen y desarrollo del Grid Computing.

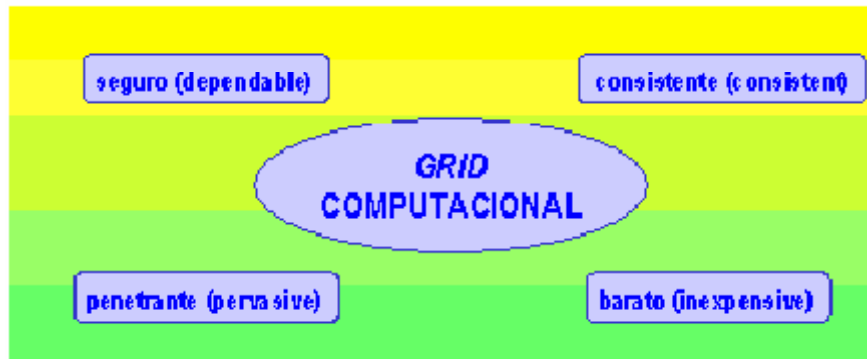


Figura 1.3: Características de un Grid Computing.

terreno de las ciencias físicas, médicas y del tratamiento de la información.

Se ha hablado mucho de lo que ofrece y de las posibilidades del *Grid Computing* pero todavía no se ha definido de manera concreta el término.

Un *Grid Computing* es una infraestructura hardware y software que suministra al que la utiliza acceso **seguro** (dependable), **consistente** (consistent), **penetrante** (pervasive) y **barato** (inexpensive), a elevadas capacidades computacionales. Ver figura 3.2 de la página 69.

El concepto de infraestructura se utiliza porque un grid es un conjunto de recursos (ciclos de CPU, datos, sensores, etc.), y todos esos recursos necesitan una interconexión de hardware y un control de software para que estén ensambladas en un grid. Esta infraestructura debe proporcionar a los usuarios un servicio seguro a todos los niveles: capacidad de cómputo, de integridad de datos, de seguridad de acceso, etc.

El servicio debe ser consistente, basado en estándares, y de esta manera el acceso y las operaciones sobre el grid estarán definidas por dichos estándares evitando la heterogeneidad. La idea de penetración no es tanto la posibilidad de acceder a cualquier recurso del grid como que el grid llega a cualquier sitio, de esta manera se asegura que una vez conectado desde cualquier punto se puede extraer del grid toda la potencia que se requiera. Por último el acceso y uso del grid debe tener un coste económico que le haga atractivo para que su utilización se universalice.

Para terminar de definir el concepto se desgana cada uno de los posibles

campos de aplicación de la tecnología grid.

El análisis lleva a definir cinco grandes áreas de trabajo determinadas por las necesidades de cálculo, espacio para el almacenamiento de los datos y tiempo de respuesta. Las áreas son:

Supercomputación Distribuida

Dentro de esta área se encuentran aquellas aplicaciones cuyas necesidades son imposibles de satisfacer en un único nodo. Estas necesidades se producen en instantes de tiempo determinados y consumen muchos recursos, por lo que se dice que son puntuales e intensivas. Ejemplo de este tipo de aplicaciones son las simulaciones, las herramientas de cálculo numérico, los procesos de análisis de datos, la extracción de conocimiento de almacenes de datos, etc.

Sistemas Distribuidos en Tiempo Real

En este tipo de aplicaciones se consideran aquellas que generan un flujo de datos a alta velocidad que debe ser analizado y procesado en tiempo real. Ejemplo de este tipo de aplicaciones son los experimentos de física de alta energía, control remoto de equipos médicos de alta precisión y precio, todos los procesos de la denominada e-Medicine, el tratamiento de imágenes para la visión artificial, etc.

Proceso Intensivo de Datos

Esta área se centra en aquellas aplicaciones que hacen un uso intensivo del espacio de almacenamiento. Las necesidades de almacenamiento de este tipo de aplicaciones desbordan la capacidad de almacenamiento de un único nodo y los datos son distribuidos por todo el grid. Además de los beneficios por el incremento de espacio, la distribución de los datos a lo largo del grid permite el acceso a los mismos de forma distribuida. Ejemplos de este tipo de aplicaciones son todos los sistemas gestores de bases de datos distribuidas.

Servicios Puntuales

En esta área, se olvida el concepto de potencia de cálculo y capacidad de almacenamiento, para centrarse en recursos que una organización puede considerar como no necesarios. De esta manera el grid ofrece a la organización esos recursos sin que la organización deba desarrollarlos por si misma. Ejemplos de este tipo de aplicaciones son aquellas que permiten acceder a hardware muy específico (equipos costosos de medida o de análisis de muestras) para la realización de labores a distancia.

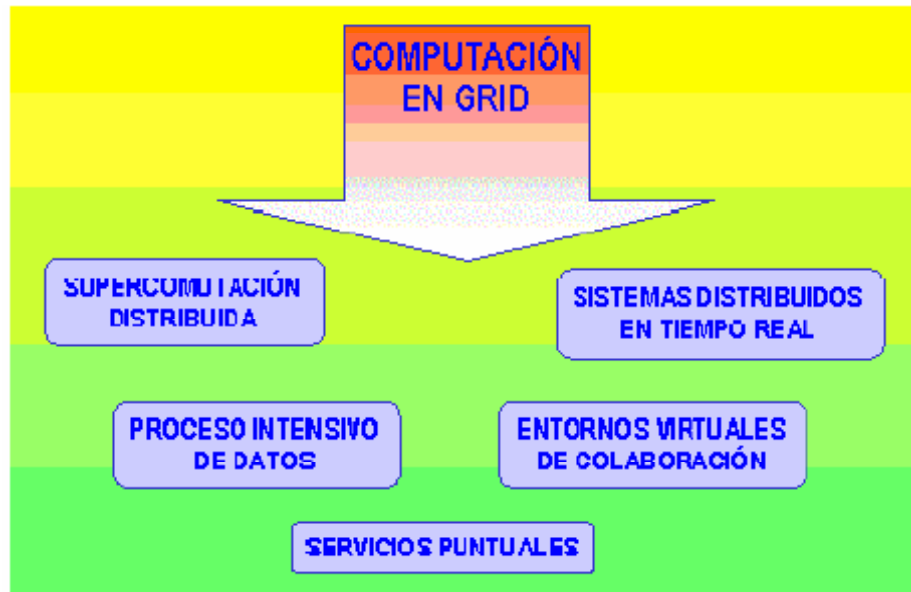


Figura 1.4: Aplicaciones de la computación en Grid.

Entornos Virtuales de Colaboración

Esta área está relacionada directamente con el concepto de *Teleinmersión*, de manera que se utilizan los enormes recursos computacionales del grid y su naturaleza distribuida para generar entornos virtuales 3D distribuidos. Ver figura 3.3 de la página 71.

1.4 Arquitectura del Grid

Es una nueva arquitectura de computación designada para direccionar las necesidades de utilidad de computación, lo que significa que la arquitectura formada por grid permite utilizar los recursos de procesamiento propios de la compañía sin tener que invertir más en adquirir capacidad extra de procesamiento. Es como colocar los procesadores existentes en la empresa en el “centro” y enviarles a ellos los datos para procesar, esto permite que los recursos sean utilizados a medida que se necesiten. [3, Brien]

Esta arquitectura se ha utilizado académicamente para resolver grandes

problemas de cómputo, como procesar datos de investigaciones, simulaciones científicas, procesamientos de datos estadísticos y simulaciones técnicas, a diferencia del grid empresarial donde se ejecutan aplicaciones reales en tiempo real y se optimizan recursos informáticos (es hacia donde está apuntando Oracle con *Grid Computing*).

Las aplicaciones, los toolkits, las APIs, los SDK, etc. que se definen para la computación en grid deben de cumplir una arquitectura general. Esta arquitectura general se articula en cinco niveles: la infraestructura, la conectividad, la gestión del recurso, la gestión de varios recursos y el nivel de aplicación, según se muestra en la figura 3.4 de la página 72.

Principalmente, la arquitectura propuesta es una arquitectura de protocolos que definen los mecanismos básicos que permiten a los usuarios y a los recursos negociar, establecer, gestionar y explotar la compartición de recursos. Es una “Arquitectura Abierta” basada en estándares que facilita la extensibilidad, la interoperabilidad, la portabilidad y la compartición de código.

De esta manera la estandarización de los protocolos permitirá estandarizar los servicios y mejorar las capacidades del grid.

En el *nivel de infraestructura* se encuentran los recursos computacionales, como son los ordenadores, los clusters, los supercomputadores, los sistemas de almacenamiento en red, las bases de datos, etc. También se incluyen en este nivel la infraestructura de la red y sus mecanismos de gestión y control. En la terminología grid la infraestructura se denomina la fábrica y suministra los componentes que serán compartidos.

El *nivel de conectividad* incluye los protocolos de comunicación y seguridad que permiten a los recursos computacionales comunicarse. Entre estos protocolos se encuentran: la pila de protocolos *TCP/IP*, el protocolo *SSL*, Certificados X.509. También se incluyen los nuevos protocolos que se encuentran en fase de estudio y que permitirán mejorar el rendimiento en las redes de alta velocidad. La “seguridad” es un punto muy importante de la computación en grid por su propia naturaleza distribuida ya que se comparten recursos entre distintas organizaciones que pueden tener distintas políticas de seguridad.

El *nivel de recurso* se centra en la gestión de un único recurso y permite tener información y control sobre el mismo. En este nivel se encuentran los protocolos que permiten obtener la información de un recurso: las características técnicas, la carga actual, el precio, etc. También se encuentran los protocolos que permiten el control del recurso: el acceso al mismo, el arranque

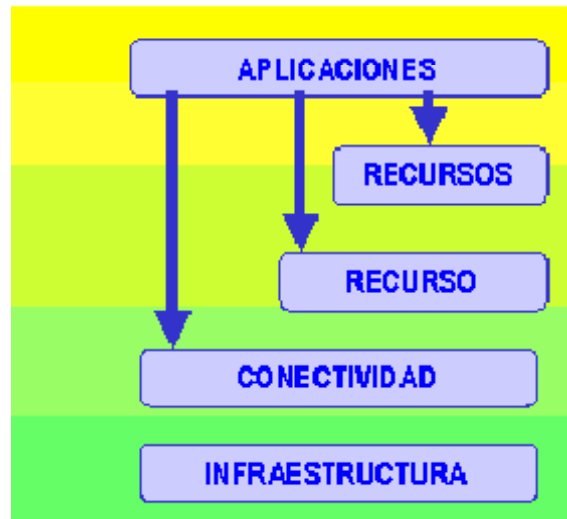


Figura 1.5: Arquitectura del grid.

de procesos, la gestión, la parada, la monitorización, la contabilidad de uso y la auditoría del recurso.

La *capa de recursos* engloba todos los servicios que permiten gestionar un conjunto de recursos. Se encuentran los servicios de directorio, que permiten localizar los recursos que son de nuestro interés; los schedulers distribuidos, que permiten asignar las tareas a cada recurso; la monitorización y diagnóstico de la ejecución de las distintas tareas en que se distribuyen la ejecución de una aplicación; la contabilidad, que permite calcular el coste de la utilización de varios recursos heterogéneos, y, el acceso a datos distribuidos, que gestiona la replicación de datos.

El “**Servicio de Scheduler Distribuido**” es una de las aplicaciones más complejas de un desarrollo grid ya que existen tres scheduler distintos: el **planificador de trabajos** (job scheduler) que intenta maximizar la cantidad de trabajo realizado (trabajos por unidad de tiempo), el **planificador de recursos** que intenta maximizar el uso de los recursos, y, el **planificador de la aplicación** que divide la aplicación en tareas, asigna los recursos para su ejecución y vigila el desarrollo de los mismos.

Los dos primeros priman la eficiencia del sistema grid, mientras que el tercero prima la eficiencia de la aplicación.

El último, *nivel de aplicación* se centra en la definición de protocolos que permitan a las aplicaciones el acceso a la infraestructura del grid a través de las distintas capas. Según el tipo de aplicación puede ser necesario conectarse a las distintas capas o acceder directamente a una de ellas, incluso directamente a la infraestructura.

1.5 Aplicaciones y Servicios en El Grid

El avance del *Grid Computing* se observa en la aplicabilidad de sus ideas en productos comerciales y los desarrollos de compañías de software basados en el estándar de grid.

1.5.1 Supercomputación

Uno de los campos donde se han centrado más los esfuerzos de compañías comerciales es en el de la *supercomputación*. Empresas con un amplio abanico de productos, ofrecen desarrollos para la instalación de grids con altas capacidades de cómputo.

Por ejemplo, HP conectó al grid del *DOE* (DOE Science Grid) una máquina Linux de 9,2 teraflops. Este grid se usa para llevar a cabo simulaciones, analizar datos y coordinar experimentos. Sobre la máquina se instala el software de *Globus* para permitir la gestión de recursos, el movimiento de datos y el control de la seguridad entre los grupos de investigación que hagan uso de él. Los usuarios se identificarán en el grid mediante las credenciales de autenticación del *GSI* (Grid Security Infrastructure), el estándar de seguridad de Globus. El supercomputador de HP será puesto a disposición del grid a través del *Globus MDS* (Monitoring and Discovery Service), un catálogo seguro de todos los recursos del grid.

GridSystems ha desarrollado el producto *InnerGrid* para la aplicación de la tecnología grid en entornos empresariales, académicos y de investigación. Este software divide datos y procesos en pequeñas unidades que se ajustan dinámicamente a los recursos conectados a la red. El sistema es multiplataforma (Windows y todos los sistemas UNIX).

Esta empresa lo que hace es aprovechar los ciclos libres de CPU aumentando la capacidad de cómputo global con un bajo costo. De esta manera los

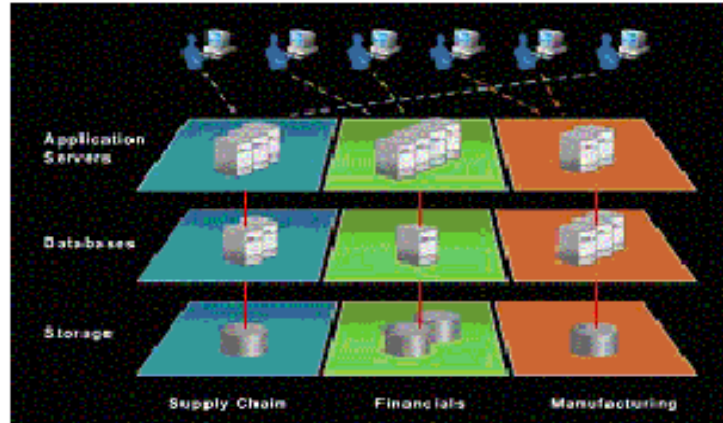


Figura 1.6: Arquitectura actual de los sistemas de información.

ciclos libres son utilizados por aplicaciones que consumen muchos recursos y necesitarían de un supercomputador o por trabajos puntuales e intensivos en una fracción de tiempo.

La instalación del producto en una empresa permite la utilización de máquinas que han quedado obsoletas y la instalación de nuevas máquinas sin que se interfiera en el trabajo de la compañía, ya que el software es fácilmente escalable.

1.5.2 Proceso Intensivo de Datos

Otra de las grandes áreas de aplicación es el *almacenamiento masivo de datos*. Dentro de esta área las nuevas versiones se centran en la aplicación de las ideas de *Grid Computing* al almacenamiento de datos.

La nueva versión parte de la arquitectura actual en la que se encuentran la mayoría de los sistemas de información. Ver figura 3.5 de la página 77.

Cada una de las capas es de uso dedicado a cada sistema de información. Evidentemente, el fallo de alguno de los componentes de los tres niveles (*Application Server*, *Database* y *Storage*) supondrá una pérdida del servicio.

Esta nueva arquitectura basada en *Grid Computing* pretende crear “granjas / factorías de almacenamiento” y servidores estándares y modulares, a las

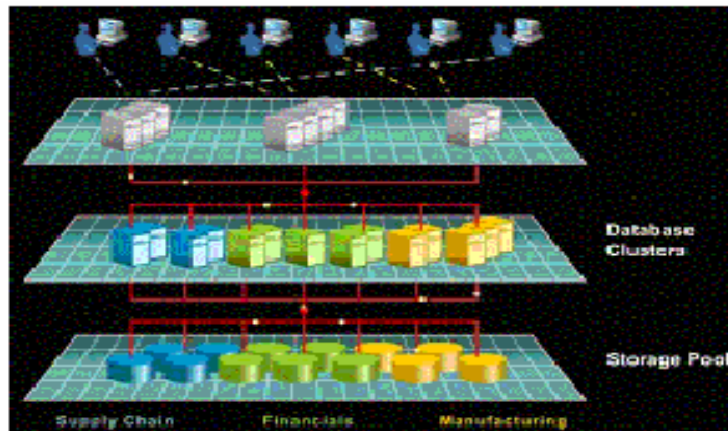


Figura 1.7: Arquitectura “Grid Computing” para sistemas de información.

que se puede añadir servidores rápidamente. En la figura 3.6 de la página 84 se detalla a grandes rasgos la arquitectura propuesta (versión Oracle 10g.).

Las características de esta arquitectura serían:

- Capacidad de balanceo de sistemas.
- Alta disponibilidad.
- Reducción de costes.
- Facilidad de administración.

La utilización de esta versión permitirá a la empresa mejorar sus prestaciones sin renunciar al comportamiento tradicional de los Sistemas de Información.

1.6 ¿Qué es el Globus?

Globus es un proyecto de investigación y desarrollo enfocado a permitir la aplicación de los conceptos de grid al campo científico.

El *Proyecto Globus* desarrolla software capaz de resolver las dificultades técnicas que aparecen al tratar de implementar un *Grid Computing*. Por ejemplo administración de recursos y datos, recursos de información, seguridad o desarrollo de aplicaciones. [8, Lopez]

El software de *Globus* es libre y está soportado por los sistemas operativos Linux, Solaris, IRIX, AIX, HPUX, True64,...

El *Globus Project* (el proyecto iniciador) es el que ha proporcionado las implementaciones estándares utilizadas por una multitud de empresas; permitiendo que los sistemas que hacen uso del grid se integren como sustento de aplicaciones empresariales.

Globus ha desarrollado el *Globus Toolkit 1.0* y después el *Toolkit 2.0* como estándares abiertos. [2, Berstis]

El *Globus Toolkit 3.0* representa un cambio significativo en la base del *Proyecto Globus*. Esta es una referencia y un estándar abierto, una implementación de la arquitectura *OGSA* que incluye una implementación de la especificación *OGSI 1.0*. El *Toolkit 3.0* es una estructura amplia de software que incluye seguridad de servicios, servicios bases, servicios de usuarios definidos, opciones de ambientes de hosting, etc.

IBM ha estado involucrada en el *Proyecto Globus* por varios años. Después de identificar varias deficiencias en las características de las últimas versiones del *Globus Toolkit*, se desarrolló y proporcionó como una base, un paquete llamado *IBM Grid Toolbox*.

El *Proyecto Globus* fue evolucionando. El equipo del *Toolbox Grid de IBM* especificó que se debió a la fuerte demanda por parte de los clientes de la empresa IBM, que manifestaban una necesidad continua de cambios en el producto, es decir un *Producto Globus* mejorado.

Este producto se convirtió en lo que hoy es el *IBM Grid Toolbox V3 for Multiplatforms V1.1*.

1.7 Estándares Abiertos

Para entender el rol desempeñado por el *Grid Toolbox de IBM*, primeramente se tienen que entender ciertos factores y discutir sobre algunos de los componentes fundamentales de los que el producto depende.

1.7.1 Web Service: Servicios Web

Un *Servicio Web* puede ser usado para construir una aplicación identificada por una dirección *Uniform Resource Locator : Localizador Uniforme de Recursos* (URL). Las interfaces y enlaces de los *Servicios Web* pueden ser definidas, descritas y descubiertas por componentes *Extensible Markup Language : Lenguaje Extensible de Marcas* (XML) y pueden soportar interacciones directas con otras aplicaciones de software usando mensajes basados en *XML* vía protocolos basados en Internet. En términos simples, un *Servicio Web* es una aplicación que se llama usando una dirección de web, pasando los parámetros en formato *XML*.

Al usar *XML*, el *Web Services Description Language : Lenguaje de Descripción de Servicios Web* (WSDL) describe una red de servicios como una colección de puntos finales que operan por medio de mensajes que contienen información ya sea orientada al proceso u orientada al documento. Para definir un punto final, se describen abstractamente operaciones y mensajes y posteriormente se limitan a un protocolo de red establecido.

Análogamente los puntos finales descritos son agrupados en puntos finales abstractos, normalmente llamado “servicios”. La funcionalidad clave de *WSDL* es permitir la descripción de productos finales y sus mensajes sin tener en cuenta los formatos de los mensajes o los protocolos de comunicación utilizados.

1.7.2 Grid Service: Servicios Grid

La tecnología de los *Servicios Grid* está basada en la *Service Oriented Architecture: Arquitectura Orientada a Servicios* (SOA) que define una arquitectura donde una aplicación se constituye de componentes independientes y cooperadores llamados “servicios”. Esos servicios construyen los bloques que utiliza un modelo de objeto para crear sistemas distribuidos abiertos y habilitar a las compañías e individuos para que creen rápidamente y en forma global sus aplicaciones disponibles para la red. [1, Aguilar]

Los mecanismos adicionales para crear y administrar *Servicios Grid* son habilitados al desarrollar un servicio nuevo que será desplegado dentro de un sistema *OGSA*. Esos mecanismos son:

Factory: Fábrica: Es una clase especial para crear dinámicamente ins-

tancias de *Servicios Grid*, código de *Servicios Grid* ejecutables y esperar por requerimientos.

Registry: Registro: Es la interfaz que habilita un conjunto de instancias de *Servicios Grid* para registrar el *Grid Service Handle: Manejador de Servicio Grid* (GSH) dentro de un servicio de registro, que permita la identificación de servicios en ese conjunto.

Discovery: Descubrimiento: Es la interfaz que permite a los clientes del *Servicio Grid* obtener información acerca de los servicios proporcionados.

Life cycle: Ciclo de vida: Se refiere a los estados de las instancias de *Servicios Grid* entre su creación y destrucción.

Service data: Datos del servicio: Es la colección estructurada de información que se asocia con una instancia de *Servicios Grid*.

Notificación: Notificación: Mecanismo por el cual una parte envía (origen de notificación) información de un cambio de estado a la parte (destino de notificación) que ha pedido ser notificada.

Reliable invocation: Invocación fiable: Técnicas que aseguran la fiabilidad de invocación de métodos en caso de que hayan sido creadas múltiples instancias con *Servicios Grid* redundantes en el espacio.

Lo importante a tener en cuenta es que el único contacto entre los *Servicios Grid* y sus usuarios es la *interfaz de servicios*. Esas interfaces de servicios son definidas por el *Lenguaje de Descripción de Servicios web* (WSDL) existente. Varias mejoras a *WSDL* han sido identificadas para requerimientos de *OGSI* y actualmente están siendo agregadas al estándar *WSDL*

1.7.3 Open Grid Services Architecture: Arquitectura de Servicios de Grid Estándar (OGSA)

El *Foro Global de Grid* fue formado para manejar las estandarizaciones en un *Grid Computing*.

La *Open Grid Services Architecture: Arquitectura de Servicios de Grid Estándar* (OGSA) del *Foro Global de Grid* representa una evolución hacia una arquitectura de sistemas basada en conceptos y tecnologías de *Servicios Web*.

Es importante destacar que *OGSA* es una arquitectura basada en los estándares existentes de *Servicios Web*, y que también se utiliza para definir muchos estándares de grid.

Los estándares de *Servicios Web* incluyen: *XML*, *SOAP* y *WSDL*.

1.7.4 Open Grid Services Infrastructure: Infraestructura de Servicios de Grid Estándar (OGSI)

El *Foro Global de Grid* promueve el desarrollo de estándares para la infraestructura de un *Grid Computing*.

OGSI se refiere a la infraestructura base sobre la cual es construida la *OGSA*. En su núcleo se encuentran las especificaciones de *Servicios Grid*, que definen la interfaz estándar y conductas de un *Servicio Grid*, armando una base de *Servicios Web*.

OGSI provee una estructura sobre la que se definen y construyen los estándares *OGSA*. Proporciona especificaciones técnicas para la implementación de cada componente de *OGSA*, usando *Servicios Grid* para definir cada interfaz. La especificación se basa en un grupo de *Servicios Web* estándar, con ciertas extensiones para *WSDL* y *XML* necesarias para los *Servicios Grid*.

OGSI define detalles tales como estabilidad de *Servicios Web*, la herencia de interfaces de *Servicios Web*, notificación asíncrona, referencias a instancias de servicios, colección de instancias de servicios y datos de estados de servicios.

El mundo de los *Servicios Web* ha reconocido las mejoras significativas logradas para *OGSA* / *OGSI* y el trabajo se encamina para incluir algunas de esas mejoras en los *Servicios Web* mismos.

1.7.5 Las tendencias futuras de los Servicios Grid

Desde el lanzamiento del *Globus Toolkit 3.0* en junio del 2003, el *GGF* y la *Alianza Globus* han trabajado estrechamente para definir mejoras al estándar.

En Enero del 2004, presentaron el *WS-Resource Framework: Entorno de Desarrollo de Recurso-WS* (WSRF), un entorno estándar para modelar y acceder a recursos que usan *Servicios Grid*.

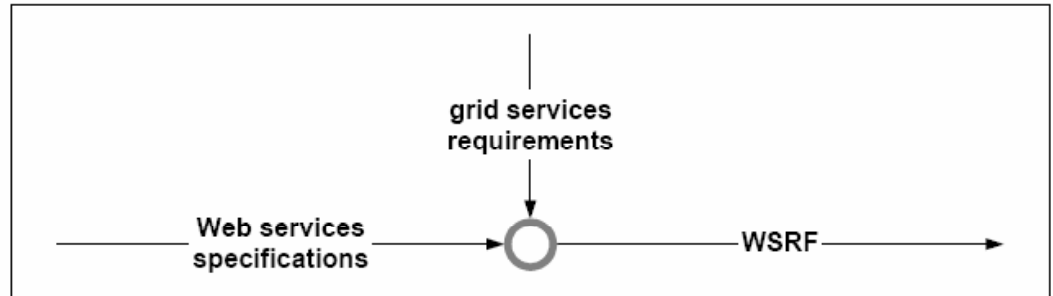


Figura 1.8: Convergencia de Servicios Web y Grid.

El *WSRF* define hacia dónde los estándares evolucionan para reunir requerimientos de *Servicios Grid* y especificaciones de *Servicios Web*. Ver figura 3.7 de la página 87.

La especificación consiste de normatizaciones separadas, cada una centrada en un área específica.

La documentación a partir de la *Open Grid Services Infrastructure: Infraestructura de Servicios Grid Estándar* alrededor del *Entorno de Desarrollo de Recursos-WS: Refactoring & Evolution Version 1.1.*, introduce las siguientes especificaciones que sirven de normas *WSRF*:

- **WS-ResourceProperties:** Especifica los estados de *Servicios Web*.
- **WS-ResourceLifetime:** Especifica los ciclos de tiempo de los *Servicios Web*.
- **WS-RenewableReferences:** Especifica las referencias y localizaciones de puntos finales de *Servicios Web*.
- **WS-ServiceGroup:** Especifica la creación y uso de grupos de *Servicios Web*.
- **WS-BaseFault:** Especifica los tipos de fallas.
- **WS-Notification:** Especifica la estructura de notificación.

1.8 El Significado de Habilitar un Grid

Hacer la habilitación de un grid significa sacar ventajas de la infraestructura virtualizada para acelerar el tiempo de procesamiento e incrementar la colaboración.

Así los estándares del grid como OGSA y el desarrollado WSRF, permitirían que una aplicación pueda ejecutarse como un *Servicio Web* en un ambiente grid mientras son usadas las ventajas de los servicios diferentes proporcionados por la infraestructura grid.

Para un programa de plataforma específica implica que la realización de ejecuciones de programas sean administradas por un producto de infraestructura grid como el Platform Computing's LSF. Para una aplicación J2EE esto puede significar su ejecución como un *Servicio Web*, o como una colección de *Servicios Web*, en el nivel más alto del middleware apto para grid, como por ejemplo el WebSphere Application Server Extended Deployment (XD) Edition.

En conclusión, para cualquier habilitación de grid con una aplicación de plataforma específica u otro tipo, esto significa tomar las ventajas de los diferentes servicios proporcionados por la infraestructura grid.

1.8.1 Estratégicamente los Servicios Web Llegan Primero

Con el tiempo, habría un requisito ineludible para habilitar un grid: las aplicaciones tendrían que ejecutarse como *Servicios Web*. Esto significa que las aplicaciones tendrán, por ejemplo, su Session EJBs expuesto como un *Servicio Web*, o tendrán que tener un artificio de *Servicio Web* (una envoltura, una interfaz, etc.) agregado al código base existente.

La asociación entre los *Servicios Grid* y *Web* es conocida oficialmente como Web Services Resource Framework: Entorno de Recursos de Servicio Web (WSRF) y que ha remplazado efectivamente a la especificación Open Grid Services Infrastructure: Infraestructura de Servicios Grid Abierta (OGSI). Como WSRF continúa evolucionando, llegará el momento en que la noción de Service Oriented Architectures: Arquitectura Orientada al Servicio (SOA) se convertirá en el nuevo paradigma de programación bajo el cual estaría el diseño de software en el futuro.

1.9 Introducción al IBM Grid Toolbox

El *IBM Grid Toolbox V3 for Multiplatforms V1.1* proporciona una instalación e integración sencilla de middleware e incorpora calidad significativa para los dos papeles más importantes del grid:

- **Diseño del grid:** Las herramientas para desarrollar y probar *Servicios Grid* y aplicaciones grid.
- **Administración del grid:** Las herramientas para organizar *Servicios Grid* y aplicaciones grid.

El *IBM Grid Toolbox V3 for Multiplatforms V1.1* implementa el estándar *OGSI* y provee las herramientas para la construcción, desarrollo, despliegue y administración de *Servicios Grid*. El *IBM Grid Toolbox* consta de los siguientes elementos:

- Un Ambiente Hosting capaz de ejecutar *Servicios Grid* y colaborar con otros participantes grid durante la ejecución de tareas extensas.
- Un conjunto de herramientas para el manejo, monitoreo y administración de *Servicios Grid* y un *Ambiente Hosting* de Grid, incluyendo una interfaz basada en la web llamada *IBM Grid Services Manager*.
- Un conjunto de APIs y herramientas de desarrollo para crear y desplegar *Servicios Grid* nuevos y aplicaciones grid.
- Un conjunto de herramientas para simplificar los procesos de instalación e integración de middleware, como el *IBM WebSphere® Application Server -Express V5.0.2*.

1.9.1 Los Objetivos del IBM Grid Toolbox

El objetivo principal del *IBM Grid Toolbox* es proveer una infraestructura común para el *Grid Computing*, manejo automático y *On Demand Solutions* (Soluciones Bajo Demanda) para la industria de Tecnologías de la Información (IT).

El desarrollo de aplicaciones y *Servicios Grid* se efectúa a un nivel muy exigente desde el punto de vista técnico y demanda de mucho esfuerzo. El *IBM Grid Toolbox* se desarrolló para clientes que ya tienen decidida la implementación de un grid, han estudiado y se conforman con las características del *Globus Toolkit 3.0*, pero están buscando un producto que se corresponda con un modelo general de opciones hardware y de software.

1.9.2 Soporte y Disponibilidad de Plataforma

El *IBM Grid Toolbox* es soportado en los siguientes ambientes:

- **Servidores IBM xSeries** que ejecutan Red Hat Enterprise Linux AS 2.1 o SUSE Linux Enterprise Server 8.
- **Servidores IBM pSeries** que ejecutan Linux SUSE Enterprise Server 8.
- **Servidores IBM pSeries** que ejecutan AIX 5L for POWER V5.2.0.10.
- **Servidores IBM iSeries** que ejecutan SUSE Linux Enterprise Server 8.
- **Servidores IBM zSeries** que ejecutan SUSE Linux Enterprise Server 8.

1.9.3 Una Visión General de los Componentes del IBM Grid Toolbox

El *IBM Grid Toolbox* es una colección de componentes que incluye al *Toolkit 3.0* del *Globus*. El siguiente resumen global de componente presenta la visión de ciertos aspectos que el *IBM Grid Toolbox* aporta a los diseñadores y administradores de *Grid Computing*:

- Procesos de instalación sencilla: El *IBM Grid Toolbox* contiene un asistente y un método de instalación, que permite una instalación sencilla del producto en el entorno de la red.
- Entorno de tiempo de ejecución de *Servicios Grid* basado en la especificación *OGSI*: Una versión embebida del *IBM WebSphere Application*

Server - Express V5.0.2 se proporciona como un contenedor de *Servicios Grid*.

- Interfaces de administración: Una interfaz llamada *IBM Grid Services Manager* que provee a los administradores una gestión sencilla del grid.
- Una infraestructura de seguridad de grid mejorada.
- Comandos de configuración y administración: Para los administradores se proveen los scripts basados en líneas de comandos, para acciones comunes.
- Herramientas de desarrollo: Se proporcionan herramientas para ayudar en la construcción, empaquetado, desarrollo y despliegue de aplicaciones y *Servicios Grid*.
- *Servicios Grid* adicionales y mejorados: IBM proporciona la funcionalidad adicional para el descubrimiento de grupo, administración de política y *Common Management Models (CMM) Services: Servicios de Modelos de Administración común*.

El *IBM Grid Toolbox* incluye una lista de *Servicios Grid* básicos que pueden ser incluidos durante la instalación o posteriormente. Se incluyen los siguientes *Servicios Grid* básicos:

- **Program Management Services:** Servicios de Administración de Programas.
- **Information Services:** Servicios de Información.
- **Data Management Services:** Servicios de administración de datos.
- **Common Management Models (CMM) Services:** Servicios de modelos de administración común.
- **Policy Services:** Servicios de política de seguridad.
- **Service Group Services:** Servicios de grupos de servicios.

En la figura 3.8 de la página 93 se detallan los macro componentes involucrados.

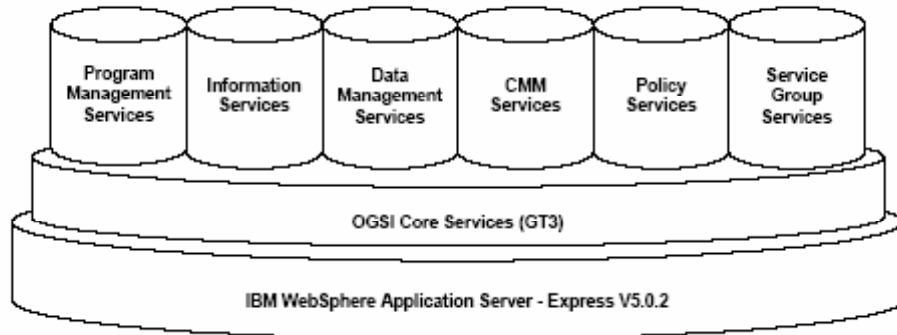


Figura 1.9: Macro componentes.

1.9.4 El Ambiente Hosting

Un *ambiente hosting* es aquel en el cual una aplicación se ejecuta. Este es el ambiente del servidor, que contiene un sistema operativo y un servidor de aplicaciones grid. El *IBM Grid Toolbox* incluye una versión embebida del *IBM WebSphere Application Server -Express V5.0.2*.

Un *ambiente de servidor* puede ser visto esencialmente como un contenedor grid ejecutándose en el interior de un motor Java (contenedor Web) o un servidor de aplicaciones *EJB* (WebSphere Application Server). La figura 3.9 de la página 94 da un ejemplo de ambiente de hosting.

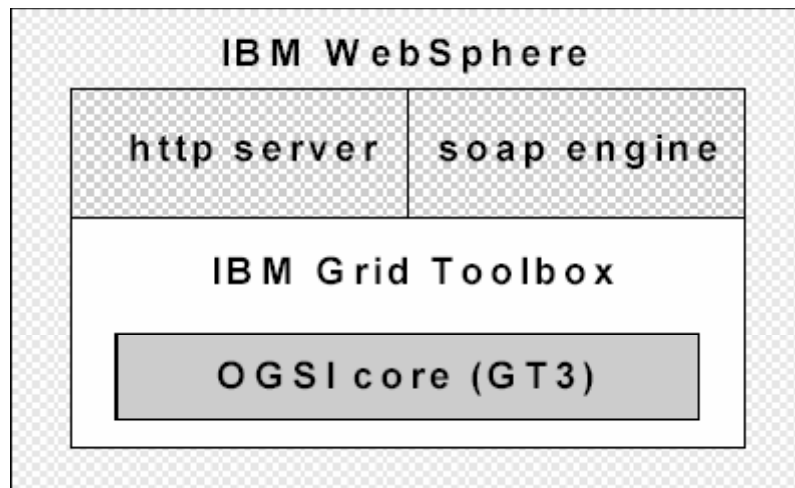


Figura 1.10: Ambiente hosting.

Capítulo 2

Principios de Grid Computing

2.1 Lo Que el *Grid Computing* Puede Hacer

Cuando se despliega un grid, se podrá encontrar con un conjunto de requerimientos del cliente.

Para que las habilidades del *Grid Computing* se adapten mejor a esos requerimientos, es útil tener presente las razones para las que se usa el *Grid Computing*.

A continuación se describen las capacidades más importantes del *Grid Computing*.

2.1.1 Aprovechamiento de los Recursos Que No Siempre Se Usan

El uso más fácil del *Grid Computing* es ejecutar una aplicación existente en una máquina diferente [1] [9] [10] [5].

La máquina en que la aplicación normalmente se ejecuta podría estar inusualmente ocupada debido a un pico inusual de actividad.

El trabajo en cuestión podría ejecutarse en otra parte en una máquina ociosa en el grid.

Hay dos requisitos previos a considerar en este apartado:

- Primero, la aplicación debe ser ejecutable remotamente.
- Segundo, la máquina remota debe encontrar cualquier hardware especial, software, o requerimientos de recursos impuestos por la aplicación.

Por ejemplo, un trabajo en lotes (batch) que consume una cantidad significativa de tiempo procesando un conjunto de datos de entrada para producir un conjunto de resultados, es quizás el uso más ideal y simple para un grid.

Si las cantidades de entradas y salidas son grandes, más análisis y planeación podrían requerirse para usar eficazmente el grid para tal trabajo.

Normalmente no tendría sentido usar un procesador de palabra remotamente en un grid, porque habría retrasos mayores y más puntos potenciales de fracaso.

En la mayoría de las organizaciones, hay grandes cantidades de recursos no siempre utilizados.

La mayoría de las máquinas del escritorio están ocupadas menos del 5% del tiempo.

En algunas organizaciones, incluso las máquinas servidoras pueden estar a menudo relativamente ociosas.

El *Grid Computing* provee una estructura para explorar estos recursos y así tener la posibilidad de aumentar la eficacia de su uso.

Los recursos del procesamiento no son los únicos que pueden ser subutilizados.

A menudo, las máquinas puede tener una enorme capacidad de disco no utilizada.

El *Grid Computing*, más específicamente, un “*Grid de datos*”, puede usarse para agregar este almacenamiento no utilizado a un almacén de datos virtuales más grandes, posiblemente configurado para lograr mejorar la calidad y confiabilidad de cualquier máquina.

Si un trabajo batch necesita leer una cantidad grande de datos, éstos datos podrían reproducirse automáticamente en varios puntos estratégicos en el grid. Así, si el trabajo debe ejecutarse en una máquina remota en el grid, los datos ya están allí y no necesitan ser movidos a ese punto remoto. Esto ofrece claros beneficios de calidad.

También, tales copias de datos pueden usarse como backups cuando las primeras copias se dañan o no están disponibles.

Otra función del grid es equilibrar mejor la utilización del recurso.

Una organización puede tener picos inesperados ocasionales de actividad que exigen más recursos.

Si las aplicaciones en el grid están habilitadas, ellos pueden moverse a las máquinas poco utilizadas durante tales picos.

De hecho, algunas aplicaciones del grid pueden migrar los trabajos parcialmente terminados.

En general, un grid puede proporcionar una manera consistente de equilibrar las cargas en una coalición de recursos más amplia.

Esto incluye la CPU, almacenamiento, y muchos otros tipos de recursos que pueden estar disponibles un grid.

La administración puede usar un grid para ver bien los modelos de uso en una organización más grande, permitiendo una mejor planificación al actualizar los sistemas, incrementar la capacidad, o retirar recursos de computing que ya no se necesitan.

2.1.2 La Capacidad de CPU Paralela

El potencial para la capacidad de CPU paralela masiva es uno de los rasgos más atractivos de un grid. Además de necesidades científicas puras, tal poder de computing está conduciendo a una nueva evolución en las industrias como el campo bio-médico, planeación financiera, exploración petrolera, etc.

El atributo común entre tales usos es que las aplicaciones se han escrito para usar algoritmos que pueden dividirse independientemente en partes de ejecución.

Una aplicación de grid intensiva de CPU puede pensarse como muchos sub-trabajos más pequeños, cada uno ejecutándose en una máquina diferente en el grid.

Si éstos sub-trabajos no necesitan comunicarse con el otro, la aplicación se vuelve más “escalable”.

Una aplicación absolutamente escalable, terminará diez veces más rápido si usa diez veces el número de procesadores.

A menudo existen barreras para perfeccionar la escalabilidad.

La primer barrera depende de los algoritmos usados para dividir la aplicación entre muchas CPUs.

Si el algoritmo sólo puede dividirse en un número limitado de partes independientes de ejecución, entonces eso forma una barrera de escalabilidad.

La segunda barrera aparece si las partes no son completamente independientes; esto puede causar contención que puede limitar la escalabilidad.

Por ejemplo, si todos los sub-trabajos necesitan leer y escribir desde un archivo común o base de datos, el acceso a ese archivo o base de datos se vuelve un factor limitante en la escalabilidad de la aplicación.

Otras fuentes de contención de inter-trabajo en una aplicación de grid paralela incluye latencias de comunicaciones de mensajes entre los trabajos, la red, las capacidades de comunicación, los protocolos de sincronización, el ancho de banda de entrada-salida, los dispositivos de almacenamiento, y latencias que interfieren con los requerimientos de tiempo real.

2.1.3 Las Aplicaciones

Hay muchos factores para considerar en una aplicación de habilitación de grid.

Se debe entender que no todas las aplicaciones pueden transformarse para ejecutarse en paralelo con un grid y lograr escalabilidad.

Además se dispone de herramientas prácticas, que los diseñadores de aplicación pueden usar para escribir una aplicación paralela.

Sin embargo, una transformación automática de aplicaciones es una ciencia nueva.

Éste puede ser un trabajo difícil y a menudo puede requerir de matemática avanzada y de talentos de programación, si incluso es posible en una situación dada.

Nuevas aplicaciones intensivas de computación se están diseñando para la ejecución paralela y éstas serán fácilmente habilitadas para su ejecución en

grid.

2.1.4 El Acceso a los Recursos Adicionales

Los recursos adicionales pueden proporcionarse en número y capacidad variable.

Por ejemplo, si un usuario necesita aumentar su ancho de banda total a Internet para implementar un mecanismo de búsqueda de minería de datos, el trabajo podría dividirse entre máquinas del grid que tienen conexiones independientes a Internet.

De esta manera, la capacidad total de búsqueda se multiplica, desde que cada máquina tiene una conexión separada a Internet.

Si las máquinas hubieran compartido la conexión en Internet, no habría habido un aumento eficaz en el ancho de banda.

Algunas máquinas pueden tener instalado el software bajo licencia costoso que los usuarios requieran.

Su trabajo puede enviarse a tales máquinas aprovechando las licencias del software.

2.1.5 Balanceo de Recursos

Para aplicaciones habilitadas, el grid puede ofrecer un efectivo balanceo de recursos mediante la planificación de trabajos de grid en máquinas con poca utilización, como se ilustra en figura 2.1 de la página 32.

Esta facilidad puede mejorar invaluablemente el manejo de picos de carga de actividad en sectores de una organización más grande. Esto puede pasar de dos maneras:

- Un pico inesperado puede ser conducido a máquinas relativamente ociosas en el grid.
- Si el grid ya se utiliza totalmente, el trabajo de prioridad más baja que se realiza en el grid debe ser suspendido temporalmente o incluso

cancelado y realizado posteriormente para dejar lugar a un trabajo de prioridad mayor.

Sin una infraestructura de grid, tales decisiones de equilibrio serían difíciles de priorizar y ejecutar.

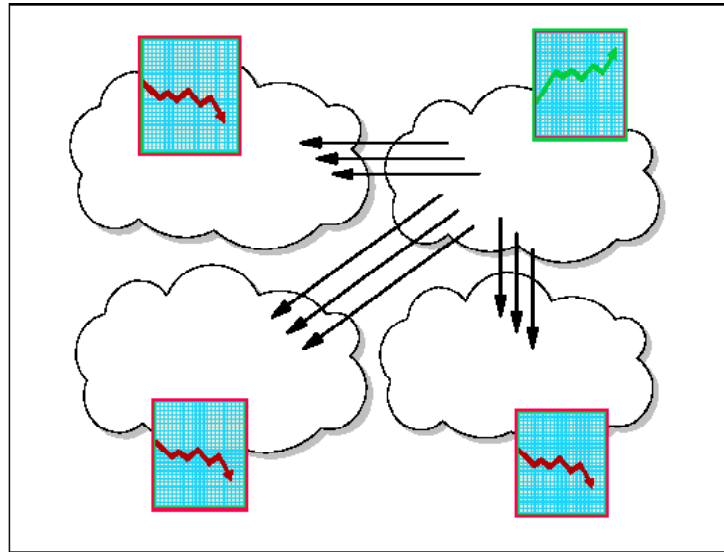


Figura 2.1: Balanceo de Recursos.

2.1.6 Confiabilidad

Los sistemas informáticos convencionales high-end usan hardware caro para aumentar la fiabilidad.

Se construyen usando chips con circuitos redundantes que brindan resultados y contienen mucha lógica para lograr una buena recuperación ante fallas de hardware.

Las máquinas también usan procesadores dobles con pluggability (conexión en caliente) para que cuando uno de ellos falla, se lo pueda reemplazar por el otro sin apagar el equipo.

Los sistemas de suministros de corriente eléctrica y de ventilación están

duplicados. Los sistemas operan con fuentes especiales de energía que pueden encender los generadores si la corriente utilizada se interrumpe. Todo esto construye un sistema fiable, pero a un gran costo, debido a la duplicación de componentes de alta fiabilidad.

Los sistemas en un grid pueden ser relativamente baratos y geográficamente dispersos.

Así, si hay algún tipo de falla, no es probable que las otras partes del grid sean afectadas (ver figura 2.2 en la página 33).

El software de gestión del grid puede automáticamente reenviar trabajos a otras máquinas del grid, cuando en una se descubre una falla.

En situaciones críticas de tiempo real, copias múltiples de trabajos importantes pueden ejecutarse en diferentes máquinas a través del grid.

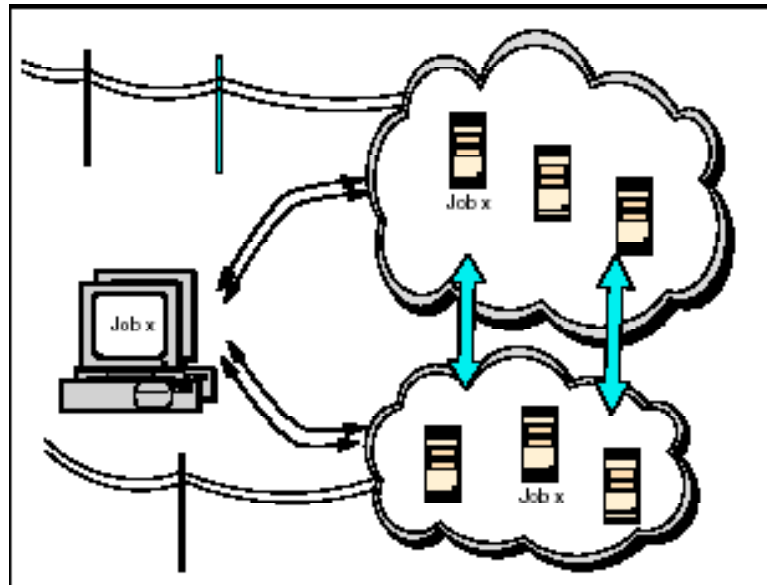


Figura 2.2: Confiabilidad en los Sistemas de Grid.

2.1.7 Administración

La meta de virtualizar los recursos en el grid y administrar más uniformemente sistemas heterogéneos crearán nuevas oportunidades para gestionar mejor una infraestructura de IT más grande, y más dispersa.

Será más fácil visualizar la capacidad y utilización, haciéndolo más fácil para los departamentos de IT y así controlar los gastos de recursos de computación en una organización más grande.

El grid ofrece administración de prioridades entre diferentes proyectos.

En el pasado, cada proyecto podría haber sido responsable de su propio hardware de recurso de IT y los gastos asociados con él.

A menudo este hardware podría no estar siempre utilizado mientras otro proyecto se encontrara en problemas, necesitando más recursos debido a eventos inesperados.

Con una visión mayor, un grid puede ofrecer controlar y manejar más fácilmente tales situaciones. Como se ilustra en la figura 2.3 de la página 35, los administradores pueden cambiar gran número de políticas que afectan en cómo las diferentes organizaciones pueden compartir o competir por los recursos.

2.2 Los Conceptos y Componentes del Grid

A continuación se introducen varios conceptos del grid, componentes, y términos más detallados.

2.2.1 Los Tipos de Recursos

Un grid es una colección de máquinas, a veces llamado “nodos”, “recursos”, “miembros”, “servidores”, “clientes”, “organizadores”, y muchos otros.

Todos ellos aportan grupos de recursos al grid. Algunos recursos pueden ser usados por todos los usuarios del grid, mientras que otros pueden tener restricciones específicas.

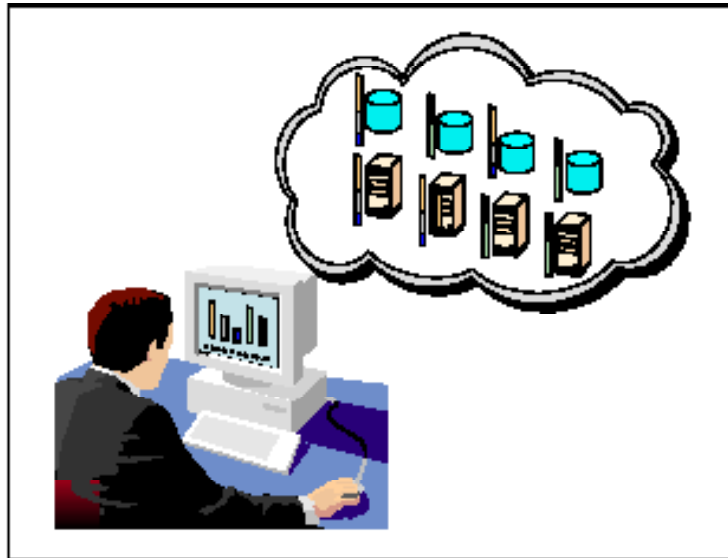


Figura 2.3: Administración en los Sistemas de Grid.

2.2.2 Computación

El recurso más común lo constituyen los ciclos de cómputo proporcionados por los procesadores de las máquinas en el grid.

Los procesadores pueden variar en velocidad, arquitectura, plataforma de software, y otros factores asociados, como la memoria, almacenamiento, y conexión.

Hay tres formas primarias para aprovechar los recursos de cómputo de un grid.

- El primero y más simple es usarlo para ejecutar una aplicación existente en una máquina disponible en el grid, en lugar de localmente.
- El segundo es para usar una aplicación diseñada para dividir su trabajo de tal manera que las partes separadas pueden ejecutarse en paralelo en los diferentes procesadores.
- El tercero es ejecutar una aplicación que precise ser ejecutada muchas veces en muchas máquinas diferentes en el grid. “Escalabilidad” es una

medida de cómo se usan eficazmente los múltiples procesadores en un grid.

Si los procesadores realizan dos veces una aplicación completa en la mitad del tiempo, entonces se dice que es absolutamente escalable.

Sin embargo, puede haber límites en la escalabilidad cuando las aplicaciones sólo pueden ser divididas en un número limitado de partes ejecutables separadamente, o si esas partes experimentan alguna otra contención para recursos de algún tipo de almacenamiento.

2.2.3 Almacenamiento

El segundo recurso más común en un grid es el almacenamiento de los datos. Un grid que proporciona una visión integrada de almacenamiento de datos es a veces llamado un “*Grid de datos*”.

Cada máquina en el grid normalmente mantiene alguna cantidad de almacenamiento para uso del grid, a veces de forma temporal.

El almacenamiento puede ser memoria unida al procesador o almacenamiento secundario, usando el disco rígido u otros medios de almacenamiento permanente.

La memoria unida a un procesador normalmente tiene un muy rápido acceso pero es volátil.

Sería mejor usarlo para guardar datos y servir como un almacenamiento temporal para las aplicaciones ejecutadas.

El almacenamiento secundario en un grid puede usarse de manera interesante para aumentar la capacidad, la calidad, el compartir, y la fiabilidad de los datos.

Muchos sistemas de grid usan sistemas de archivo de red montables tales como, Andrew File System (AFS), Network File System (NFS), Distributed File System (DFS), o el General Parallel File System (GPFS). Estos ofrecen diferentes grados de calidad, seguridad y fiabilidad.

La capacidad puede ser incrementada usando el almacenamiento en múltiples máquinas con un sistema de archivo unificado.

Cualquier archivo individual o base de datos pueden expandirse a varios dispositivos de almacenamiento y máquinas, eliminando las restricciones de máxima capacidad, a menudo impuestas por los sistemas de archivo de los sistemas operativos.

Un sistema de archivo unificado también puede mantener un sólo espacio uniforme para el almacenamiento del grid.

Esto facilita a los usuarios referenciar los datos que residen en el grid, sin considerar su ubicación exacta.

De manera similar, el software especial de base de datos puede “federar” un conjunto de bases de datos individuales y archivos para formar bases de datos más amplias y más comprensivas.

Como se observa en la fig. 2.4 de la pág. 38 el **Data Striping** significa escribir o leer sucesivos datos desde o para diferentes dispositivos físicos, solapando el acceso para un mejor rendimiento; asimismo, con técnicas adicionales se aumenta la fiabilidad.

Los sistemas de archivo más avanzados en un grid pueden duplicar automáticamente conjuntos de datos, para proporcionar redundancia para mayor fiabilidad y aumento del rendimiento.

Un scheduler de grid inteligente puede ayudar a seleccionar los dispositivos de almacenamiento apropiados para guardar datos, basados en patrones de uso.

Los trabajos pueden ser programados más cerca de los datos, preferentemente en las máquinas directamente conectadas a los dispositivos del almacenamiento que guardan los datos requeridos.

2.2.4 Las Comunicaciones

El crecimiento rápido en la capacidad de comunicación entre las máquinas hace que el *Grid Computing* sea práctico, comparado con el limitado ancho de banda disponible cuando la computación distribuída emergió por primera vez.

Por consiguiente, no debe ser una sorpresa que otro recurso importante de un grid sea la capacidad de comunicación de datos. Esto incluye las comunicaciones dentro del grid y fuera de él.

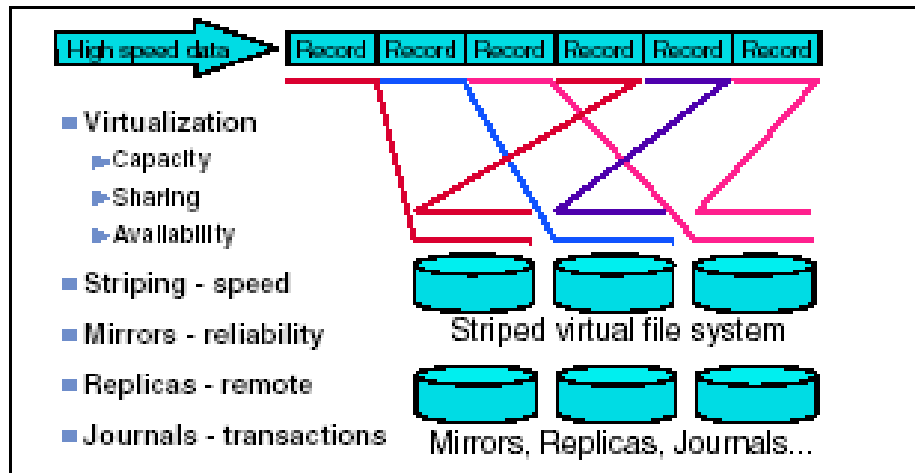


Figura 2.4: Almacenamiento en el Grid.

Las comunicaciones dentro del grid son importantes para enviar trabajos y sus datos requeridos hacia puntos dentro del grid.

Algunos trabajos exigen procesar una cantidad grande de datos y no siempre los mismos pueden residir en la máquina que ejecuta el trabajo.

El ancho de banda disponible para tales comunicaciones frecuentemente puede ser un recurso crítico que puede limitar la utilización del grid.

El acceso de comunicación externo a Internet, por ejemplo, puede ser valioso al construir motores de búsqueda.

Las máquinas en el grid pueden tener conexiones externas a Internet, en adición a las conexiones entre las máquinas del grid. Cuando estas conexiones no comparten la misma ruta de comunicación, se agregan al ancho de banda disponible, para acceder a Internet.

A veces se necesitan rutas de comunicación redundantes para manejar mejor fallas potenciales en la red y el excesivo tráfico de datos.

En algunos casos se deben utilizar redes de altas prestaciones para satisfacer demandas de trabajos que transfieren cantidades muy grandes de datos.

Un sistema de administración de grid puede mostrar mejor la topología del grid y puede resaltar los cuellos de botella de comunicación. Esta información

puede usarse para planificar las actualizaciones del hardware.

2.2.5 El Software y las Licencias

El grid puede tener el software instalado que puede ser demasiado caro para instalar en cada máquina de éste.

Usando un grid, los trabajos que requieren este software son enviados a máquinas particulares en las cuales este software está instalado.

Cuando las tarifas de licencia son significantes, esta aproximación puede ahorrar gastos importantes para una organización.

Algún arreglo de licencia de software permite instalarlo en todas las máquinas de un grid pero puede limitar el número de instalaciones que pueden usarse simultáneamente en cualquier momento.

El software de administración de licencias registra cuántas copias coexistentes de éste están usándose y previene que se ejecute un número mayor en un tiempo dado .

Los schedulers de trabajo de grid pueden configurarse para tener en cuenta las licencias de software, opcionalmente balanceándolas contra otras prioridades o políticas.

2.2.6 El Equipo Especial, Capacidades, Arquitecturas, y Políticas

Las plataformas en el grid tendrán a menudo diferentes arquitecturas, sistemas operativos, dispositivos, capacidades, y equipos.

Cada uno de estos ítems representa un tipo diferente de recurso que el grid puede usar con criterio para asignar trabajos a las máquinas.

Mientras algún software puede estar disponible en varias arquitecturas, por ejemplo PowerPC y x86, tal software es a menudo diseñado para ejecutar sólo un tipo particular de hardware y sistema operativo. Tales atributos deben ser considerados al asignar recursos en el grid.

En algunos casos, el administrador de grid puede crear un nuevo tipo de recurso artificial que será usado por el schedulers para asignar el trabajo

según el tipo de política u otras restricciones. Por ejemplo, algunas máquinas pueden diseñarse para sólo ser usadas para investigación médica y otras para sólo participar en el grid si no se usan para propósitos militares.

2.2.7 Los Trabajos y las Aplicaciones

Aunque varias clases de recursos en el grid pueden compartirse y usarse, normalmente son accedidos vía la ejecución de una “aplicación” o “trabajo”.

Normalmente se usa el término “aplicación” como el nivel más alto de un porción de trabajo en el grid. Sin embargo, a veces el término “trabajo” se usa equivalentemente.

Las aplicaciones pueden dividirse en cualquier número de trabajos individuales, como se ilustra en la fig. 2.5 de la pág. 40; aquéllas, a su vez, pueden dividirse en sub-trabajos.

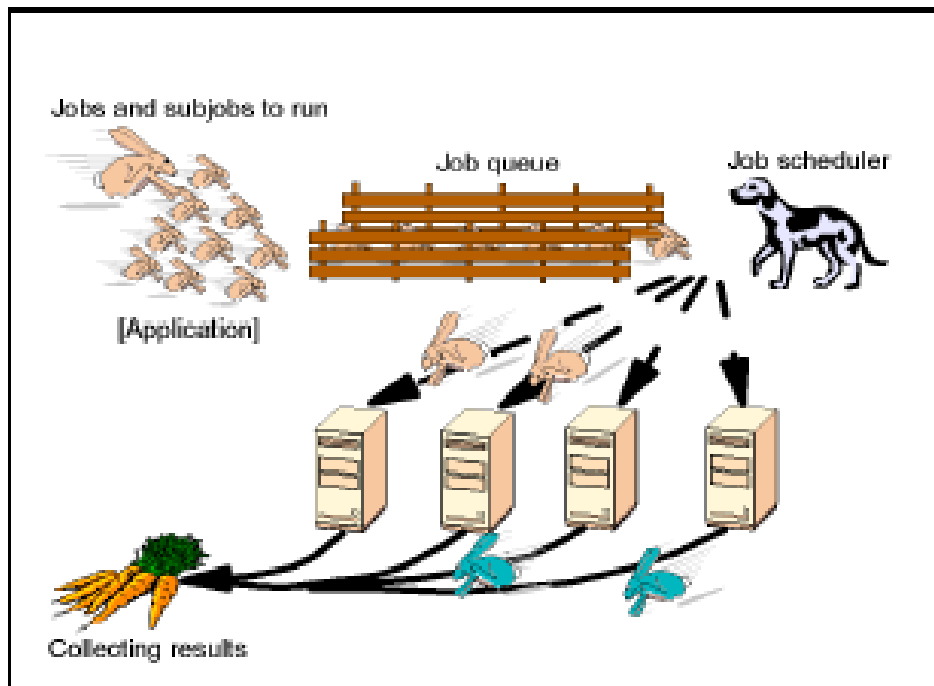


Figura 2.5: Trabajos y Aplicaciones del Grid.

2.2.8 Scheduling, Reservación, y Barrido

El sistema del grid es responsable de enviar un trabajo a una máquina dada para ser ejecutado.

En el más simple sistema de grid, el usuario puede seleccionar una máquina conveniente para realizar su trabajo y luego ejecutar una orden de grid, que envía el trabajo a la máquina seleccionada.

Los sistemas de grid más avanzados incluirían un trabajo de “scheduler” de algún tipo que, automáticamente, encuentre la máquina más apropiada en la cual ejecutar cualquier trabajo dado, el cual estaría esperando ser ejecutado.

Los schedulers reaccionan a la disponibilidad actual de recursos en el grid.

El término “scheduling” no debe ser confundido con la “reservación” de recursos para mejorar la calidad de servicio.

En un “scavenging” (barrido) de un sistema grid, cualquier máquina que se vuelve ociosa (sin trabajo local que ejecutar) informaría su estado al nodo de administración del grid. Este nodo de administración asignaría a esta máquina el próximo trabajo que se satisfecería por los recursos de la misma.

El “scavenging” normalmente es llevado a cabo de manera que no obstruya al usuario.

Si la máquina se pone en estado de “ocupada” con el trabajo local no-grid, el trabajo del grid normalmente se suspende o se demora. Esta situación crea de alguna manera tiempos de terminación imprevisibles para los trabajos del grid, aunque no molesta a las máquinas donantes de recursos.

Para crear una conducta más predecible, las máquinas del grid se dedican a menudo al grid y no pueden ser retiradas del mismo para realizar trabajos externos al grid. Esto permite al scheduler computar el tiempo aproximado de realización para un conjunto de trabajos, cuando sus características de ejecución son conocidas.

En el próximo paso, los recursos del grid pueden “reservarse” por adelantado para un conjunto de trabajos designados. Esto se hace para reunir fechas tope y garantizar la calidad de servicio.

Cuando las políticas lo permitan, los recursos reservados de antemano podrían ser recogidos para ejecutar trabajos de menor prioridad, cuando no están

ocupados durante un período de reservación.

Así, varias combinaciones de scheduler, reservación y barrido, pueden ser usados para utilizar un grid más completamente.

El scheduling y reservación es bastante directo cuando sólo un tipo de recurso, normalmente, CPU, está involucrado. Sin embargo, las optimizaciones de grid adicionales pueden ser logradas considerando más recursos en proceso de reservación y planificación.

Por ejemplo, sería deseable asignar trabajos de ejecución a las máquinas más cercanas a los datos que estos trabajos requieren.

Esto reduciría el tráfico de la red y posiblemente reduciría los límites de escalabilidad.

La planificación óptima, considera múltiples recursos, es por ello que se la considera un problema matemático difícil. Por consiguiente, tales schedulers pueden usar heurísticas. Estas heurísticas son reglas que se diseñan para mejorar la probabilidad de encontrar la mejor combinación de schedulers de trabajo y reservaciones para perfeccionar el rendimiento o cualquier otra métrica.

2.3 Intragrid a Intergrid

Como se presenta en fig. 2.6 de la pág. 44, el grid más simple consiste en sólo unas pocas máquinas, todas de la misma arquitectura de hardware y el mismo sistema operativo, conectadas en una red local. Este tipo de grid usa sistemas homogéneos, así hay menos consideraciones y pueden ser usados sólo para experimentar con el software de grid.

Las máquinas usualmente están en un departamento de una organización, y sus usos como un grid pueden no requerir una política especial de seguridad. Debido a que las máquinas tienen la misma arquitectura y sistema operativo, elegir software de aplicación para dichas máquinas es generalmente sencillo. Esto podría ser llamarlo una aplicación de “cluster” en lugar de “grid”.

El siguiente avance sería incluir las máquinas heterogéneas. En esta configuración, hay más tipos de recursos disponibles. El sistema grid puede incluir algunos componentes de scheduling. También puede lograrse el compartimiento de archivos, usando los sistemas de archivo de red.

Las máquinas que participan en el grid pueden incluir uno de los departamentos de la misma organización. Tal modelo de grid también será llamado “Intragrid”.

Cuando el grid se extiende a muchos departamentos, las políticas pueden requerirse para indicar cómo el grid debe usarse. Por ejemplo puede haber políticas, para qué tipos de trabajo permite el grid y cuántas veces.

Puede haber también una priorización por departamento o por tipos de aplicaciones que deben tener acceso a los recursos del grid.

Los datos sensibles en un sólo departamento pueden necesitar ser protegido contra el acceso de trabajos que se ejecutan para otros departamentos.

Las máquinas del grid especializadas pueden ser agregadas para aumentar la calidad de servicio, en lugar de depender completamente de los recursos de “barrido”.

El grid puede crecer geográficamente en una organización que tiene los medios en diferentes ciudades.

Las conexiones de comunicaciones dedicadas pueden usarse entre estas facilidades y el grid.

En algunos casos, VPN tunneling u otras tecnologías pueden usar Internet para conectar las diferentes componentes de la organización. La seguridad aumenta una vez que los límites de cualquier facilidad se superan.

El grid puede crecer para ser jerárquicamente organizado para reducir la contención implicada por el control central, aumentando la escalabilidad.

Un grid puede crecer, cruzar los límites de la organización y puede usarse para colaborar en los proyectos de interés común. Esto se conoce como un “Intergrid” (como se ve en la fig. 2.7 de la pág. 2.7).

Habitualmente se requieren los niveles más altos de seguridad en esta configuración para prevenir posibles ataques y espionajes.

El Intragrid ofrece la posibilidad de comerciar recursos a un público más amplio.

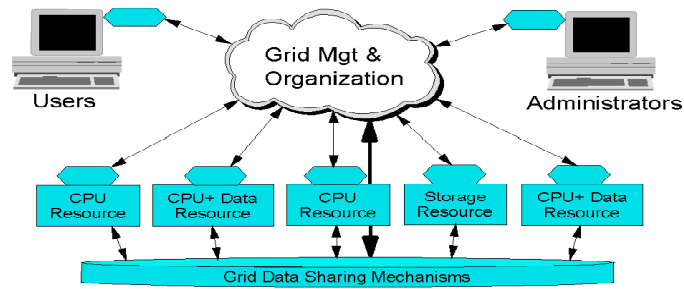


Figura 2.6: Un Grid más Simple.

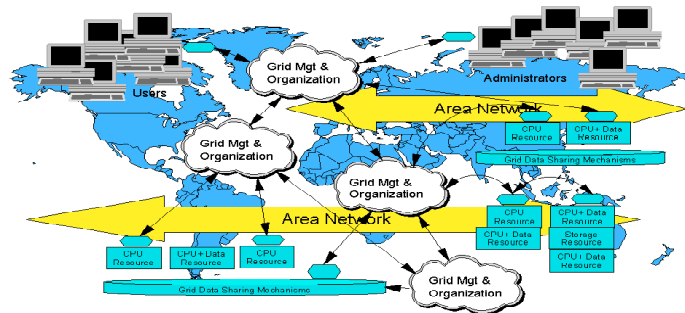


Figura 2.7: Un Intergrid más Complejo.

2.4 Construcción del Grid

Un grid ad hoc puede instalarse por unos programadores en su tiempo libre, pero como el grid crece, y los usuarios se vuelven más dependientes de él para el trabajo misión-crítico, un grado de planeamiento es esencial.

Es mejor entender los requerimientos de la organización y escoger las tecnologías del grid que mejor se adapten a estos requerimientos.

Esta sección ya se han discutido sobre algunas consideraciones de planificación y componentes del grid que cumplen con los requerimientos.

2.5 Planificación del Despliegue

El uso de un grid nace a menudo de una necesidad de incrementar recursos de algún tipo. Por ello una de las primeras consideraciones son el hardware disponible y cómo se conecta vía una LAN o WAN.

Luego, una organización puede querer agregar el hardware adicional para aumentar las capacidades del grid. Es importante entender las aplicaciones a ser usadas en él.

Sus características pueden afectar las decisiones de cómo escoger y configurar el hardware y sus datos conectados.

2.5.1 Seguridad

La seguridad es un factor muy importante; planear y mantener un *Grid Computing* convencional distribuido dónde el compartimiento de datos reduce el volumen de la actividad.

En un grid, las máquinas miembro se configuran para ejecutar los programas en lugar de sólo mover datos. Esto es lo que hace un grid potencialmente fértil para virus y programas de Caballo de Troya.

Por esta razón, es importante entender qué componentes exactamente del grid deben afianzarse para detener rigurosamente cualquier clase de ataque.

Además, es importante comprender los problemas involucrados al autenti-

car usuarios y ejecutar adecuadamente las responsabilidades que esto implica.

2.5.2 Organización

Las consideraciones tecnológicas son importantes al desplegar un grid. Sin embargo, los problemas comerciales y de organización pueden ser igualmente importantes.

Es primordial entender cómo los departamentos en una organización interactúan, operan y contribuyen al todo.

Frecuentemente hay barreras construídas entre los departamentos y proyectos para proteger sus recursos, en un esfuerzo de aumentar la probabilidad de éxito.

Sin embargo, volviendo a preocuparse por algunas de estas relaciones, se puede encontrar, que más compartimiento de recursos a veces puede beneficiar a la organización entera.

Por ejemplo, un proyecto que se encuentra detrás del scheduler y encima del **budget** no puede admitirse el lujo de conseguir los recursos requeridos para resolver el problema.

Un grid agregaría una medida de seguridad, proporcionando un margen extra de capacidad de recurso, necesitado para finalizar el proyecto. De manera similar, un proyecto en sus fases tempranas cuando los recursos de computadora no están utilizándose totalmente, puede donarlos a otros proyectos que los necesitan.

Un grid también ofrece la habilidad para la administración de organización, y ver así el cuadro de prioridad más grande y reaccionar más rápidamente cambiando la utilización del recurso, prioridades, y políticas.

2.6 Componentes del Software Grid

En esta sección se presenta algunos de los componentes importantes que deben discutirse antes de diseñar una arquitectura de *Grid Computing*.

2.6.1 Componentes de Administración:

Cualquier sistema de grid tiene algunos componentes de administración.

Primero hay un componente que guarda , la muestra de los recursos disponibles para el grid, y qué usuarios son miembros de él. Esta información se usa principalmente para decidir dónde deben asignarse los trabajos del grid.

Segundo, hay componentes de medida que determinan, las capacidades de los nodos y su proporción de utilización actual en cualquier momento dado.

Tal información se usa también para determinar la robustez del grid, advirtiéndolo al personal sobre los problemas como, los paros, congestión, u **over commitment**.

Esta información también es usada para determinar modelos de uso globales y estadísticas, así como para registrar y responder por el uso de recursos del grid.

Tercero, el software de administración del grid avanzado puede manejar muchos aspectos automáticamente.

Esto se conoce como “autonomic computing”, o “computing de recuperación” orientada. Este software se recuperaría automáticamente de diversos tipos de fracasos del grid y paros, encontrando maneras alternativas de conseguir el trabajo procesado.

2.6.2 Software Servidor

Cada máquina que contribuye con recursos típicamente necesita conectarse como un miembro del grid e instalar algún software que maneje el uso de los recursos del grid. Habitualmente, alguna clase de proceso de identificación y autenticación deberá realizarse antes de que una máquina pueda unirse al grid.

Un certificado de mando puede usarse para establecer la identidad de la máquina donante, también como los usuarios y el grid mismo.

Algunos sistemas de grid proporcionan su propio login, mientras otros dependen de los sistemas operativos procedentes para la autenticación del usuario.

En el último caso, un sistema de mapeo “ID de usuario” puede ser necesario para unir adecuadamente los derechos del usuario a las diferentes máquinas. Esto es típicamente mantenido de forma manual por un administrador del grid.

Él determina qué ID de usuario puede poseer qué máquina del grid y qué base de datos o registros.

De esta manera, cuando los trabajos del grid se presentan a diferentes máquinas para un usuario, donde el ID del usuario de la máquina local es usado para determinar sus derechos.

En algunos sistemas de grid, es posible unir el mismo sin ninguna autenticación especial. Y en otros, es posible para cualquier usuario presentar trabajos al grid.

Tales sistemas pueden ser convenientes para set up pero no para despliegues mas grandes, debido a problemas de seguridad serios que ellos ocasionarían.

El sistema de grid hace que la información sobre los recursos recientemente agregados se encuentre disponibles a lo largo de él.

La máquina donante normalmente tendrá alguna clase de monitor que determine o mida cuan ocupada está y la proporción o cantidad de recursos utilizados.

Esta información es “bubbled up” al software de dirección del grid y usada para scheduler de acuerdo con esos recursos.

En un sistema barrido de basura, esta información le dice al software de dirección del grid, cuándo la máquina está ociosa y disponible para el trabajo.

Es importante que el software instalado en una máquina dada pueda aceptar un trabajo ejecutable desde un sistema de administración de grid y ejecutarlo.

El software de gestión de grid debe ser capaz de recibir el archivo ejecutable o seleccionar el propio, desde copias preinstaladas en la máquina donante. Éste lo ejecuta y el resultado es enviado devuelta al cliente.

Implementaciones mas avanzadas pueden dinamicamente ajustar la prioridad de un trabajo en ejecución suspenderlo y continuar después, o controlarlo con la posibilidad de continuar su ejecución en una máquina diferente.

Estos tipos de acciones pueden ser necesarios para responder a problemas de nivelación de carga, prioridad o cambios de política en el grid.

2.6.3 Software de Sumisión

Normalmente cualquier máquina miembro del grid puede usarse para realizar los trabajos e iniciar sus requerimientos.

Sin embargo, en algunos sistemas de grid, esta función se lleva a cabo como un componente separado, instalado en “nodos de sumisión” o “clientes de sumisión”. Cuando un grid se construye usando recursos especializados en lugar de “recursos scavenged” el software de sumisión separado es normalmente instalado en la estación de trabajo.

2.6.4 Administración del Grid Distribuido

Los grid más grandes normalmente pueden tener una topología organizacional o jerárquica uniendo las conexiones topológicas. Es decir, que las máquinas localmente conectadas con una LAN forman un cluster de máquinas. Éste puede organizarse en una jerarquía que consiste en cluster de cluster.

El trabajo involucrado en la administración del grid se distribuye para aumentar su escalabilidad. Programa un trabajo subordinado directamente a la máquina para ejecutarlo.

En cambio el trabajo se envía a un programador de menor nivel que ocupa un conjunto de máquinas. Este se ocupa de la asignación de máquinas específicas.

De la misma forma se distribuye la colección de información estadística.

Los cluster de menor nivel reciben la información de actividad, la agregan y las envían desde las máquinas individuales a los nodos de dirección de niveles más altos en la jerarquía.

2.6.5 Schedulers

La mayoría de los sistemas de grid incluyen alguna clase de software de programación de trabajo.

Este software localiza una máquina en la cual ejecutar un trabajo de grid que ha sido presentado por un usuario.

En los más simples de los casos, puede asignar ciegamente los trabajos en forma round-robin a la próxima máquina cumpliendo con los requerimientos del recurso. Sin embargo hay ventajas al usar un scheduler más avanzado.

Algunos schedulers llevan a cabo un sistema de prioridad de trabajo. Esto a veces se hace usando varias colas de trabajo, cada uno con una prioridad diferente. Cuando las máquinas del grid están disponibles para ejecutar los trabajos, éstos son tomados en orden de prioridad.

Varias clases de políticas son llevadas a cabo usando los programadores.

Las políticas pueden incluir los varios tipos de restricciones en los trabajos, los usuarios, y recursos. Puede haber una política de que restrinja la ejecución de trabajos de grid en ciertos momentos del día.

Los schedulers normalmente reaccionan a la carga inmediata del grid. Ellos usan la información de medida, sobre la utilización actual de máquinas para determinar cuáles no están ocupadas, antes realizar un trabajo.

Los schedulers pueden organizarse en una jerarquía.

Por ejemplo, un meta-scheduler puede enviar un trabajo a un scheduler del cluster o a otros scheduler de menor nivel en lugar de a una máquina individual.

Los schedulers más avanzados supervisarán el progreso de trabajos programados conduciendo el flujo de trabajo global. Si los trabajos se pierden debido al sistema o paros de la red, un buen scheduler reenviaría automáticamente el trabajo a otra parte.

Sin embargo, si un trabajo parece estar en un loop infinito y alcanza una interrupción máxima, entonces no deberan reprogramarse tales trabajos.

Típicamente, los trabajos tienen diferentes tipos de códigos de realización, algunos de los cuáles son convenientes para el reenvío y otros no.

Reservar los recursos en el grid de antemano se realiza mediante un “sistema reservación”.

Este es primero, un sistema basado en un calendario para reservar los recursos con períodos de tiempo específico y así evitar que otros reserven el

mismo recurso al mismo tiempo.

Este también debe poder quitar o suspender trabajos que podrían estar ejecutándose en cualquier máquina o recurso, cuando el período de la reservación es alcanzado.

2.6.6 Las Comunicaciones

Un sistema de grid puede incluir un software para ayudar a los trabajos a comunicarse entre sí.

Por ejemplo, una aplicación puede dividirse en un gran número de sub-trabajos. Cada uno de estos es un trabajo separado en el grid. Sin embargo, la aplicación puede ser llevada a cabo por un algoritmo que requiere que los sub-trabajos se comuniquen alguna información entre ellos.

Los sub-trabajos necesitan ser capaces de localizar otros sub-trabajos específicos, establecer una conexión de comunicaciones con ellos, y envían los datos apropiados.

El estándar abierto Message Passing Interface (MPI) y cualquier otra variación es a menudo incluida como parte del sistema de grid sólo para este tipo de comunicación.

2.6.7 Observación, Dirección, y Decisión

Se mencionó que los schedulers reaccionan ante las cargas actuales en el grid.

Habitualmente, el software servidor incluirá algunas herramientas que midan la carga actual y la actividad en una máquina dada usando, las facilidades de un sistema operativo o por medición directa.

Este software es a veces llamado “sensor de carga”.

Algunos sistemas de grid proporcionan los medios para implementar sensores de carga personalizada en lugar de CPU o recursos del almacenamiento.

Tal información de medición es útil no sólo para programar sino también para descubrir el uso de patrones globales en el grid.

Las estadísticas pueden mostrar tendencias que pueden señalar la necesi-

dad de un hardware adicional. También, la información de medición sobre trabajos específicos pueden ser recolectados y usarse para predecir mejor los requerimientos de recurso del trabajo que se ejecutara la próxima vez.

Cuanto mejor sea la predicción más eficiente será el trabajo del grid.

La información de medición también puede ser guardada con propósitos de contabilidad, para formar la base, la ejecución de un recurso de grid, o para manejar las prioridades de forma más justa.

La información también puede ser mostrada en varios formatos para visualizar mejor actividad y utilización del grid.

2.7 Usar un Grid: Perspectivas de Usuario

2.7.1 Conectar e Instalar el Software de Grid

Un usuario se conecta primero como un usuario de grid, e instala el software en su propia máquina.

Él también puede conectar su máquina como un servidor en grid.

Conectarse en él puede requerir la autenticación con propósitos de seguridad.

El usuario efectivamente establece su identidad con un certificado de autoridad. Esto no debe hacerse solamente vía Internet.

La autoridad hace que un certificado este disponible para las necesidades del software para asegurar así la identidad de un usuario de grid y sus requerimientos.

Pasos similares pueden ser requeridos para identificar la máquina donante.

El usuario tiene la responsabilidad de guardar y proteger sus credenciales de grid.

Una vez el usuario y/o la máquina se autentican, el software del grid se proporciona al usuario para instalar en su máquina con propósitos de usar el grid, así como servir a éste.

Este software puede ser automáticamente pre-configurado por el sistema de

administración del grid para conocer la dirección de comunicación, los nodos de dirección en el grid e información de identificación del usuario o máquina.

De esta manera, la instalación puede realizarse haciendo un click con un mínimo de interacción requerida por parte del usuario.

En las instalaciones del grid menos automatizadas, puede pedirse al usuario que identifique el nodo de dirección del mismo y posiblemente otra información de configuración.

Él puede escoger limitar los recursos servidos al grid, las veces que su máquina es utilizada por dicho grid, y otras restricciones relacionadas con la política del sistema.

El usuario también puede que necesite informar al administrador del grid que IDs de usuario son suyos en otras máquinas que existen en el grid.

2.7.2 Registrarse En el Grid

Para usar el grid la mayoría de los sistemas exigen al usuario que se registre en un sistema usando el ID de usuario que se matricula en el grid.

Otros sistemas de grid pueden tener su propio login de grid ID separado del sistema operativo.

Un registro del grid es usualmente más conveniente para los usuarios de grid. Este elimina los problemas de conexión de ID entre las diferentes máquinas.

También hace que el grid se parezca más a una gran computadora virtual en lugar de una conjunto de máquinas individuales.

Por ejemplo, Globus lleva a cabo un modelo de login que mantiene al usuario registrado por una cantidad específica de tiempo, aun cuando él se desconecta y vuelve al sistema operativo o cuando la máquina es reanudada.

Una vez registrado el usuario puede solicitar al grid y realizar los trabajos.

Algunas aplicaciones del grid permiten algunas funciones de solicitud si el usuario no está conectado en él o aun cuando el usuario no este registrado (**enrolled**) en el grid.

2.7.3 Solicitar y Realizar Trabajos.

El usuario normalmente realizará algunas preguntas para verificar y para ver cuan ocupado está el grid, para ver cómo sus trabajos realizados están progresando, y para buscar los recursos en él.

Los sistemas del grid habitualmente proporcionan las herramientas de línea de orden así, como interfaces gráficas de usuario (GUIs) para las preguntas.

Las herramientas de línea de orden son especialmente útiles cuando el usuario quiere escribir un documento que automatiza una sucesión de acciones.

Por ejemplo, el usuario podría escribir un documento para que busque un recurso disponible, realice un trabajo para él, mire el progreso del trabajo, y presente los resultados cuando el trabajo ha terminado.

La realización del trabajo normalmente consiste en tres partes, aun cuando hay sólo una orden requerida.

Primero, algunos datos de entrada y posiblemente programas ejecutables o la ejecución de un archivo de documento, se debe enviar a la máquina para ejecutar el trabajo; enviar la entrada se llama “escenario de los datos de entrada”.

Alternativamente, pueden pre-instalarse los datos y archivos del programa en las máquinas del grid o hacerlas accesibles vía un montable sistema de archivo de red de computadoras.

Cuando el grid consiste en máquinas heterogéneas, puede haber archivos de programa ejecutables múltiples, cada uno compilado en diferentes plataformas de máquinas en el grid.

Un rasgo bueno proporcionado por algunos sistemas grid es registrar estas versiones múltiples del programa, para que el sistema de grid pueda escoger automáticamente una versión que se conecte correctamente a la máquina de él, la cual ejecutará el programa.

Algunas tecnologías de grid requieren que el programa y los datos de la entrada primero se procesen o incluyan (“**wrapped**”) de alguna manera por el sistema de grid. Esto puede hacerse para agregar controles de ejecución protectores alrededor de la aplicación o simplemente recolectar todos los archivos de datos en uno.

Segundo, el trabajo se ejecuta en la máquina del grid.

El software de grid que se ejecuta en la máquina donante, ejecuta el programa en un proceso a favor del usuario.

Puede usar a un ID de usuario común en la máquina o puede usar ID de usuario propio del usuario , dependiendo de que tecnología de grid se usa.

Algunos sistemas de grid llevan a cabo un “sandbox” protector alrededor del programa, para que este último no cause ninguna daño a la máquina servidora si encuentra un problema durante la ejecución. Pueden restringirse derechos para acceder archivos y otros recursos en la máquina del grid.

Tercero, se envían de vuelta los resultados del trabajo al cliente.

En algunas aplicaciones, los resultados intermedios pueden ser vistos por el usuario que realizó el trabajo.

En algunas tecnologías de grid que no envían de vuelta automáticamente los datos del resultado al usuario, deben ser explícitamente enviados al usuario, posiblemente usando un sistema de archivo conectado a una red de computadoras.

Los documentos también son útiles para realizar una serie de trabajos, para una aplicación espacial de parámetro, por ejemplo.

Algunos problemas del cómputo consisten en una búsqueda de resultados deseados basados en algunos parámetros de entrada. La meta es encontrar los parámetros de entrada que producen los mejores resultados deseados.

Para cada parámetro de entrada, un trabajo separado se ejecuta para encontrar el resultado a ese valor.

La aplicación entera consiste en muchos trabajos como estos, los cuales exploran los resultados un gran número de valores de parámetro de entrada.

Los documentos normalmente se usan para enviar los sub-trabajos, cada uno, recibiendo sus propios valores de parámetro particular.

Las entradas de parámetro a veces pueden ser más complejas, que solo un número. A veces un conjunto de datos de entrada diferente representa el parámetro “input”.

Los documentos ayudan a automatizar la gran variedad de problemas del

estudio espacial de parámetros.

Para entradas espaciales de parámetro más simple, algunos productos de grid proporcionan un GUI para realizar una serie de sub-trabajos, cada uno con valores de parámetro de entrada diferentes.

Cuando hay un gran número de sub-trabajos, el trabajo requerido para recolectar los resultados y producir el resultado final es normalmente realizado por un sólo programa, mientras se ejecuta en la máquina en el momento de realización del trabajo.

Si hay un gran número de sub-trabajos requeridos para una aplicación, el trabajo de compilar los resultados también podría ser distribuido.

Por ejemplo, el sub-trabajo que envíe más sub-trabajos al grid sería responsable de recolectar y agregar los resultados que estos obtuvieron.

2.7.4 Configuración de Datos

Los datos obtenidos por los trabajos del grid simplemente pueden organizarse adentro y a fuera del sistema de grid.

Sin embargo, dependiendo de su tamaño y el número de trabajos, este puede potencialmente agregarse a una gran cantidad de tráfico de los datos. Por esta razón, algunas ideas se dan acerca de cómo obtener el mínimo movimiento de los datos en el grid.

Por ejemplo, si va a haber un número muy grande de sub-trabajos ejecutándose en la mayoría los sistemas del grid para una aplicación que se ejecutara repetidamente, los datos que ellos usan pueden copiarse para cada máquina y residir hasta la próxima vez que se ejecute una aplicación. Esto es mejor que usar un sistema del archivo conectado a una red de computadoras para compartir estos datos, porque en tal sistema de archivo, los datos serían efectivamente movidos desde una ubicación central cada vez que la aplicación se ejecute.

Esto es cierto a menos que el sistema de archivo lleve a cabo una exclusiva copia o duplique los datos automáticamente.

Hay muchas consideraciones en la planificación eficiente de la distribución y compartimiento de datos en un grid.

Este tipo de análisis es necesario para grandes trabajos para así poder utilizar bien el grid y no crear cuellos de botella innecesarios.

2.7.5 Monitoreo del Progreso y Recuperación

El usuario puede solicitar al sistema de grid, para ver cómo su aplicación y sus sub-trabajos están progresando.

Cuando el número de sub-trabajos crece, se vuelve mas difícil de listarlos a todos en una ventana gráfica. En cambio puede haber simplemente un sólo gráfico grande de barra que puede mostrarlos.

Es más difícil para el usuario decir si algún sub-trabajo particular no se ejecuta apropiadamente.

Un sistema de grid, junto con su scheduler de trabajo, proporciona a menudo algún grado de recuperación para sub-trabajos que fallan.

Un trabajo puede fallar debido a un:

- Error de Programación: El trabajo se detiene alguna parte con alguna falla de programa.
- Fallo de Hardware : La máquina o dispositivos que se usan dejan de trabajar de alguna manera.
- Interrupción en las Comunicaciones: Un camino de comunicación a la máquina ha fallado o ha sido cargado excesivamente con otro tráfico de datos.
- Lentitud Excesiva: El trabajo podría estar en un loop infinito o el progreso de trabajo normal puede ser limitado por otro proceso que se ejecuta con una prioridad mayor.

No siempre es posible determinar automáticamente si la razón del fracaso de un trabajo es debido a un problema con el diseño de la aplicación o si es debido a los fracasos de varios tipos en la infraestructura del sistema del grid.

Los schedulers se diseñan a menudo para categorizar los fracasos del trabajo de alguna manera y automáticamente rehacer éstos para que ellos tengan éxito, mientras se ejecuten en otra parte del grid.

En algunos sistemas, el usuario está informado sobre cualquier fracaso del trabajo y debe decidir si emitir una orden para intentar reejecutar los trabajos fallados.

Las aplicaciones de grid pueden diseñarse para automatizar el monitoreo y recuperación de sus propios sub-trabajos usando las funciones proporcionadas por las interfaces de programación (APIs) de aplicación del software de sistema de grid.

2.7.6 Reservar Recursos

Para mejorar la calidad de un servicio, el usuario puede acordar reservar un conjunto de recursos por adelantado para su uso exclusivo o de mayor prioridad.

Una analogía del sistema de calendario puede usarse aquí. Tal sistema de reservación también puede usarse junto con el hardware planeado o eventos de mantenimiento de software, cuando el recurso afectado no están disponible para el uso del grid.

En un grid de recolección de residuos, no es posible reservar las máquinas específicas de antemano. En cambio, los sistemas de dirección de grid pueden asignar un fragmento más grande de su capacidad para una reservación dada para permitir la posibilidad de que algunos de los recursos se vuelvan no disponibles.

Esto debe hacerse junto con herramientas que han perfilado la capacidad de trabajo de grid para tener las estadísticas fiables sobre la habilidad de dicho grid y así dar la reservación.

2.8 Usar un Grid: La Perspectiva de Un Administrador

2.8.1 Planeación

El administrador debe entender los requerimientos de la organización del grid, para elegir mejor las tecnologías del grid, que satisfagan esos requerimientos. Las siguientes secciones describen brevemente los pasos que el administrador

debe tomar para manejar el grid.

Se sugiere que se comience por desplegar un pequeño grid, para aprender sobre su instalación y dirección, antes de tener que confrontar los problemas más complicados involucrados con un grid grande.

2.8.2 Instalación

Primero, el sistema de grid seleccionado debe instalarse en un conjunto de máquinas apropiadamente configuradas. Estas máquinas usando redes con amplitud suficiente para otras máquinas en el grid.

Es importante entender los escenarios de fracaso para el sistema de grid dado, de manera que este continúe operando aun cuando cualquiera de las máquinas de dirección falle de alguna manera.

Las máquinas deben configurarse y conectarse para facilitar los escenarios de recuperación.

Cualquier base de datos crítica u otros datos esenciales para guardar la muestra de los trabajos, los miembros del grid, y las máquinas, éstos deben tener posibilidad de backups.

Además, los certificados claves públicos deben tener una copia de seguridad y las claves privadas deben guardarse en un lugar seguro inaccesible por otros.

Después de la instalación, el software del grid puede necesitar ser configurado para la dirección de la red local e IDs.

El administrador normalmente requerirá el acceso base a las máquinas conductoras en el grid.

En algunos sistemas de grid, él necesitará también el acceso **base** a las máquinas servidoras requeridas para instalar el software también en aquellas. El software a ser instalado en las máquinas servidoras puede necesitar ser personalizado que para pueda encontrar automáticamente las máquinas de dirección (management) del grid e incluir las claves públicas pre-instaladas para el grid.

Este software puede proporcionarse para servidores potenciales en un FTP o el servidor equivalente o sea disponible en los medios de comunicación físicos.

Una vez, que el grid es operacional, puede haber software de aplicación y datos que deben ser también instalados en las máquinas servidoras. Este software puede tener restricciones específicas de autorización, las cuales deben entenderse y adherirse. Algunos sistemas de grid incluyen herramientas para ayudar con la dirección de la autorización de un grid amplio. Este puede ayudar seguir las reglas de autorizaciones y explotar eficazmente esas autorizaciones.

2.8.3 Matriculación de Dirección de Servidores y Usuarios

Una tarea continua para el administrador del grid es dirigir a los miembros del grid, a las máquinas servidoras de recursos y los usuarios.

Los usuarios pueden organizarse como grupos de proyecto.

El administrador es responsable de controlar los derechos de los usuarios en el grid.

Las máquinas donantes pueden tener derechos de acceso que requieren también la dirección.

La ejecución de trabajos de grid en las máquinas donantes puede hacerse bajo un especial ID usuario de grid a beneficio de los usuarios que realizan los trabajos.

Los derechos de éstos IDs de usuario de grid deben ser apropiadamente puestos para que los trabajos no permitan el acceso a las partes de la máquina donante en las cuales el usuario no está registrado.

Cuando los usuarios se conectan al grid, su identidad debe establecerse positivamente y debe entrar en el certificado de autoridad.

El usuario y sus credenciales de certificado deben ser agregadas a la lista del usuario usando el software apropiado para el sistema de grid desplegado.

En algunos casos, el administrador debe propagar la información del usuario a varias o todas las máquinas.

También, cuando el sistema de grid depende principalmente del sistema operativo, para el login del usuario, el administrador puede necesitar para agregar entradas para perfilar al usuario del grid específicos sistema operativo de IDs de usuario en las máquinas donantes.

2.8. USAR UN GRID: LA PERSPECTIVA DE UN ADMINISTRADOR 61

Una actividad de registro similar normalmente se exige para registrar máquinas donantes en el grid.

La identidad de máquina se establece y registra con el certificado la autoridad.

El administrador del grid debe estar en acuerdo con el administrador de la máquina donante sobre el IDs de usuario, software, derechos de acceso, y cualquier restricción de política.

El administrador debe introducir las credenciales de identificación de la máquina, direcciones, y características de recurso usando el software apropiado para registrar la máquina servidora del grid.

En algunos casos, el administrador puede necesitar propagar esta información manualmente a otras máquinas en el grid.

Los procedimientos correspondientes para quitar usuarios y máquinas deben ejecutarse por el administrador.

2.8.4 Certificado de Autoridad

Es crítico asegurar los más altos niveles de seguridad en un grid ya que éste se diseña para ejecutar un código y no sólo para compartir datos. Así, este puede ser fértil para virus, los Trojan horses, y otros ataques, si el sistema de grid se compone de alguna forma.

El certificado de autoridad es para mantener una fuerte seguridad del grid.

Una organización puede escoger usar un certificado de autoridad externo u operar uno ella misma.

Usted debe poder confiar en el certificado de autoridad para adherirse estrictamente a sus responsabilidades.

Las responsabilidades primarias de un certificado de autoridad son:

- Identificar positivamente las entidades que piden los certificados.
- Emitir, quitar, y archivar los certificados.
- Proteger al servidor del certificado de autoridad.

- Mantener un namespace de nombres únicos para los dueños del certificado.
- El servidor firma los certificados a aquéllos que necesiten autenticar entidades.
- Actividad de Registracion.

Brevemente, un certificado de autoridad se basa en un sistema de encriptación de clave pública.

En este sistema, se generan las claves de a pares, una clave pública y una privada.

Cualquiera de las dos puede usarse para encriptar algunos datos, tal que la otra se necesita descifrarlo.

La clave privada es guardada por el dueño y nunca revelada a nadie, y la pública es dada a cualquiera.

Un certificado de autoridad es usado para mantener estas claves públicas y garantizar a quienes ellas pertenecen.

Cuando un usuario usa su clave privada para encriptar algo, el receptor usa la correspondiente clave pública para descifrarlo.

El receptor sabe que solo la clave pública puede descifrar el mensaje correctamente.

Sin embargo, cualquiera podría interceptar este mensaje y podría descifrarlo porque cualquiera puede conseguir la clave pública original.

Si el creador encripta doblemente el mensaje con su clave privada y la clave pública del destinatario, una segura conexión de comunicación se forma.

El receptor usa su clave privada para descifrar el mensaje y luego usa la clave pública del remitente para la segunda decriptación.

Ahora el destinatario sabe que si el mensaje se descifra apropiadamente, sólo el remitente lo podría haber enviado y además, este sabe que sólo el receptor al que se dirige podrá descifrarlo.

Lo bueno de todo esto es que nadie tiene que llevar en forma segura una clave de encriptación del remitente al receptor, como debe hacerse para los

sistemas de encriptación convencionales, y cualquier alteración en la comunicación se revela.

Así, este puede ser fértil para virus y los Trojan horses, y otros ataques si el sistema de grid se compone de alguna forma.

Un intercambio similar se usa para obtener la clave pública de alguien desde el certificado de autoridad, para que el usuario sepa que él ha recibido una clave pública inalterada para el usuario deseado.

2.8.5 Administración de Recursos

Otra responsabilidad del administrador es manejar los recursos del grid.

Esto incluye establecer los permisos para los usuarios del grid para usar los recursos, así como ruteo del uso del recurso y llevar a cabo un correspondiente sistema de contabilidad o registro.

El uso de estadísticas es útil en la identificación de direcciones en una organización que puede requerir la adquisición de hardware adicional, la reducción de hardware en exceso para reducir los costos, y ajustes de prioridades y políticas para lograr una utilización más justa o lograr mejor las metas globales de una organización.

Algunos componentes del grid, normalmente, los schedulers de trabajo, tienen provisiones para reforzar prioridades y políticas de varios tipos.

Es responsabilidad del administrador configurarlos para el mejor logro de los objetivos de la organización global.

Los administradores de licencia de software pueden usarse en un grid para controlar la utilización apropiada. Éstos pueden configurarse para trabajar con schedulers de trabajo y para priorizar el uso de las licencias limitadas.

2.8.6 Compartir Datos

Para grid pequeños, el compartir datos puede ser bastante fácil, usando los sistemas de archivo existentes, conectados a una red de computadoras, a bases de datos, o protocolos estándar de transferencia de datos. Cuando un grid crece y los usuarios se vuelven dependientes de cualquiera de los almacenes de

almacenamiento de datos, el administrador debe considerar los procedimientos para mantener copias de seguridad y réplicas, para mejorar así la realización.

Todos lo que se refiere a la dirección de recursos se aplican a los datos en el grid.

2.9 Usar Un Grid:Una Perspectiva del Diseñador de la Aplicación

Las aplicaciones de grid pueden categorisarse en una de las siguiente tres categorías:

- Aplicaciones que no se habilitan para usar los procesadores múltiples pero pueden ejecutarse en diferentes maquinas.
- Aplicaciones que ya son diseñadas para usar los procesadores múltiples de un conjunto del grid.
- Aplicaciones que necesitan ser modificadas o reescritas para aprovecharse mejor un grid.

La última categoría es de interés para diseñadores de aplicación de grid. Ellos encontrarán una necesidad por las herramientas para depurar y medir la conducta de aplicaciones del grid. Tales herramientas basadas son aun nuevas.

Puede ser útil para diseñadores configurar un grid pequeño para poder usar los depuradores en cada máquina, para así controlar y mirar los funcionamientos detallados de las aplicaciones. Ya que el proceso de depuración puede desviar ciertas precauciones de seguridad, puede no siempre ser eficaz para permitir tal depuración en un grid de producción.

Globus es más bien un equipo de herramientas del diseñador (toolkit) para construir componentes de grid, en lugar de un el sistema de grid comprensivo. Este posee los componentes básicos necesarios para construir nuevos medios para administrar funcionamientos del grid, medición, reparación, y depuración de aplicaciones.

Las herramientas en surguimiento que conforman las interfaces de la Arquitectura de Servicios de Grid Abierta (OGSA)serán utilizables en varios sistemas de grid del vendedor.

2.10 El Presente y el Futuro del Grid

El toolkit de Globus es un juego de herramientas útil para construir un grid. Su fuerte es un modelo de alta seguridad, con una provisión para jerárquicamente coleccionar datos sobre el grid, así como facilidades básicas para implementar un simple, world-spanning grid.

El Globus crecerá con el tiempo promete extenderse a través del trabajo de muchas organizaciones que están desarrollando sus capacidades.

La mayoría de los sistemas de grid incluyen algún schedulers de trabajo, pero como áreas más amplias de medición de grid, habrá una necesidad para más meta-schedulers que puedan manejar colecciones de clusters diversamente configurados y los grid más pequeños.

Estos schedulers evolucionarán para programar bien los trabajos, considerando múltiples recursos en lugar de sólo utilización de CPU.

Ellos también extenderán su alcance para llevar a cabo una mejor calidad de servicio, usando reservaciones, redundancia, e historia de perfiles de trabajos y la calidad de un grid.

Hoy, los sistemas de grid todavía están en las fases tempranas para proporcionar un almacenamiento y compartir datos virtuales de manera fiable, bien realizada, y automáticamente recuperables.

Se verán productos que colocan esta tarea en un conjunto de grid, creando datos de todo tipo, y logrando una mejor calidad, integración con la programación, fiabilidad, y capacidad.

El computing autónomo tiene el objetivo de hacer el trabajo del administrador más fácil, automatizando varias tareas complicadas involucradas en la administración de un grid. Éstos incluyen identificación de problemas en tiempo real y una rápida iniciación de acciones correctivas, antes de que ellos dañen seriamente el grid.

La Arquitectura de Servicios de Grid Abierta (OGSA) es una norma abierta a la base de todas éstas mejoras de los grid futuros.

OGSA estandarizará las interfaces del grid que serán usadas por los nuevos schedulers, agentes de computing átomos y cualquier otro servicio a ser desarrollado por el grid. Este lo hará más fácil para reunir los mejores productos

de varios vendedores, aumentando el valor del *Grid Computing* global.

2.10.1 ¿Qué No Puede Hacer el *Grid Computing*?

El grid no es una bala color de plata que puede tomar cualquier aplicación y ejecutarla mil veces más rápido sin la necesidad de comprar más máquinas o software.

No toda aplicación es conveniente o capaz de ejecutarse en un grid.

Algunos tipos de aplicaciones simplemente no pueden ser puestas en forma paralela.

En otros casos, puede tomar una gran cantidad de trabajo para modificarlos y así lograr un movimiento más rápido.

La configuración de un grid puede que afecte la calidad, fiabilidad, y seguridad de una infraestructura de organización de computing.

Por todas estas razones, es importante entender cómo ha evolucionado el grid hoy y qué características tendrá en un futuro distante.

Capítulo 3

Definición del Caso Para el Grid

Se debe tener en cuenta que el *Grid Computing* que se prefiera adaptar brindaría los resultados que, como desarrollador de software y como cliente se pretendería alcanzar. El grid ayuda a las organizaciones al ahorrar dinero perfeccionando la utilización de sus inversiones, así como también a los gerentes a incrementar los ingresos y beneficios al acelerar los tiempos de procesamiento de las aplicaciones y mejorar la administración de la infraestructura de aplicación.

El objetivo principal del *Grid Computing* es proveer toda una infraestructura, en la que sea posible el manejo automático y las *On Demand Solutions* (soluciones bajo demandas).

3.1 Todo Acerca de On Demand

El grid no se presenta aisladamente. El grid no se trata de la adopción de tecnología para crear tecnología. No es algo que debería considerarse sólo por ser una visión nueva. El grid es parte de la amplia visión *On Demand*. La figura 3.1 de la página 68 ilustra la manera en que se da la evolución de ambientes operativos tradicionales a ambientes operativos *On Demand*.

En 1998 se dijo al mundo que el Modelo de Programación J2EE era la

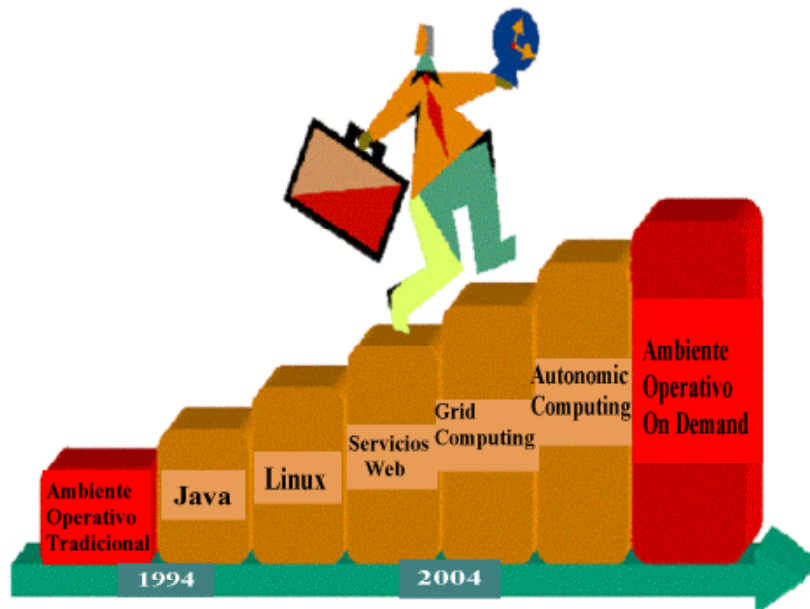


Figura 3.1: Camino a la Visión On Demand.

manera de llegar al futuro. En aquellos tiempos también se dijo que el Linux global era otra de las maneras de llegar. Se entrenó a la gente para que pudieran utilizar Linux; se crearon roadmap, talleres, artículos y otros métodos para conseguir que Linux sea el sistema operativo de opción, o por lo menos una plataforma de soporte para los productos.

Al mismo tiempo, se empezó a hablar de los Servicios Web, se crearon los roadmap para líneas enteras de productos para que las aplicaciones pudieran correr como Servicios Web.

En el 2005, el J2EE se establece firmemente como un modelo de programación maduro y fiable. Los productos de J2EE continúan evolucionando perfectamente y el Linux se ha adaptado a todas las plataformas de hardware.

La apariencia de los Servicios Web continúa evolucionando y está madurando muy rápidamente, sobre todo después de la llegada del Web Services Resource Framework: Entorno de Recursos de Servicios Web (WSRF) como un sustituto para la especificación Open Grid Services Infrastructure: Infraestructura de Servicios Grid Abierta (OGSI).



Figura 3.2: Los Principios en On Demand.

3.1.1 ¿Por Qué Prestar Atención a On Demand?

Se ve a *On Demand* como una estrategia enfocada a la transformación de procesos comerciales, para lograr ventajas del ambiente operativo obteniendo flexibilidad comercial y simplificación de la infraestructura tecnológica de información. Todo esto dentro de un contexto financiero flexible y opciones de entrega. La figura 3.2 de la página 69 ilustra tal integración.

El objetivo de alinear estos tres elementos (Business Processes, Business Transformation y *On Demand* Operating Environment) es simple: permitir a los negocios convertirse a *On Demand* soportando las necesidades de los clientes y de esta manera flexibilizar y responder en tiempo real.

3.1.2 Ambiente Operativo

Son tres los elementos del ambiente operativo *On Demand*: Integración, Automatización y Virtualización.

- **Integración:** Se refiere a la integración de datos, aplicaciones, dispositivos, procesos y personal de una empresa.
- **Automatización:** El tope de las proporciones de utilización, el costo de infraestructura y la complejidad están incrementando. Esta tendencia

seguiría aumentando tanto como el costo de la tecnología y su proliferación misma. Hay un fuerte apoyo en la automatización de la administración de infraestructura, seguridad y control desarrollado completamente con capacidades autónomas (autonomic computing). Como los ambientes son virtualizados, hay grandes oportunidades para desplegar herramientas de administración para automatizar estos ambientes.

- **Virtualización:** La virtualización crea una asociación de recursos que pueden ser administrados, accedidos y modificados más eficientemente; desde un almacenamiento específico y con tecnologías de virtualización de servidor en niveles de compartimiento para SAN y soluciones grid.

3.1.3 Todo Acerca de la Evolución

El ambiente operativo *On Demand* es el plato Petri donde la transformación de los negocios se efectúa. Es la fundación que permite a los negocios convertirse a negocios *On Demand*. Así como el mercado evoluciona para determinar la manera en que se efectúa la comercialización, *On Demand* evoluciona para establecer la manera en que la informatización de los negocios es alcanzada. Ver Figura 3.3 de la página 71.

Cuando se menciona a la productividad organizacional se está haciendo referencia al funcionamiento de ambientes integrados, donde el software no reside en lugares dispersos de automatización, si no que es parte de un ambiente integrado donde todo se interconecta.

Si lo que se pretende es que los productos operen en ambientes operativos *On Demand*, se necesitará cambiar la mitología de negocio y venta del Software. Si la decisión apunta a la implementación, se va a tener que adaptar. Y el primer cambio en la adopción tiene que ver con la constitución, el diseño e integración de tecnología.

El desafío de convertirse a centros *On Demand* implica estar listos para evolucionar y adaptarse a los cambios, como probablemente se hizo años atrás cuando se decidió adoptar por J2EE y por los estándares abiertos. El cambio es bueno y las innovaciones están orientadas a las tecnologías que facilitarían las actividades de negocios cotidianas de las compañías haciéndolas más integradas, autónomas y virtualizadas.

Se tiene que observar a los clientes. Lo que se implemente deberá tener cordura para ellos. Las personas que compran el producto; ¿Quieren conver-

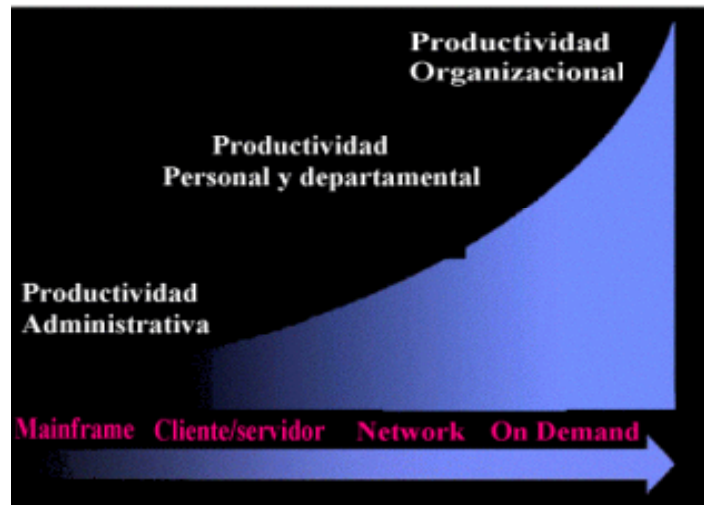


Figura 3.3: De Administrativo a la Productividad Organizacional.

tirse a *On Demand*? Si la respuesta es si, entonces los productos deberán también orientarse a *On Demand*.

3.1.4 La Propuesta Clave

Si se es proveedor de una industria que puede beneficiarse con la visión *On Demand*, es probable que la mayoría de los representantes de esa industria se decidan por *On Demand*. Por consiguiente tendrá que, o bien seguir con las mismas ideas o cambiar su enfoque, en aquellos casos en que lo requieran. La propuesta clave está en “crecer con el mercado y beneficiarse de él”.

Convirtiéndose a la visión *On Demand* se está listo para la evolución y adaptación a las nuevas tecnologías que facilitarían a los clientes las actividades de negocios cotidianas. El mensaje es que se apunte a la capa de virtualización del ambiente *On Demand*. Se necesitará centrarse en la virtualización como una manera de proporcionar las herramientas a los clientes para optimizar su infraestructura actual, a través de las tecnologías como el *Grid Computing*.

Las Opciones

Como se expreso anteriormente, la habilitación del grid es un ejercicio

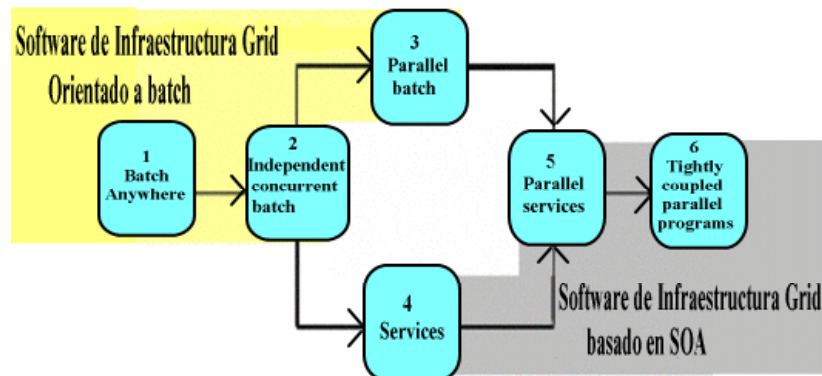


Figura 3.4: Estrategias de Habilidad de Grid para Arquitecturas Orientadas al Servicio y en Lotes (Batch).

de transformación de software que puede significar desde la integración de un trabajo a reescribir el código por completo. La magnitud del esfuerzo dependerá de lo que se pretenda hacer.

Existen estrategias para minimizar el esfuerzo de construir un grid habilitando código existente. También aquellas que apunta a minimizar el esfuerzo de diseñar y escribir nuevo código para habilitar un grid desde su origen. Lo más importante de recordar es que se tiene todas las opciones y oportunidades para elegir.

3.2 Los Grado de Adopción de Grid

3.2.1 Las Estrategias Para la Habilidad de Grid

Se considerarán seis estrategias y dos contextos. Ver la figura 3.4 de la página 72.

Seis Grados de Grid

Existen seis modos diferentes en que una aplicación puede correr en un Grid:

- *Batch Anywhere (Lote en cualquier lugar)*: En este tipo de habilitación se da cierta independencia a la situación. En este caso, la aplicación se ejecutaría como un trabajo batch de instancia simple sobre cualquier nodo del grid.
- *Independent Concurrent Batch (Lotes concurrentes independientes)*: La concurrencia puede ser alcanzada a través de este tipo de habilitación de grid. Es posible tener varias instancias independientes de la ejecución de un mismo trabajo corriendo en nodos diferentes del grid. Las instancias del trabajo pueden asociarse a diferentes usuarios (uno por instancia) o varias instancias pueden ser ejecutadas en representación de un único usuario.
- *Parallel Batch (Lote paralelo)*: Esta es una manera simple de lograr el paralelismo. Hay una instancia de un trabajo dividiéndose entre todos los nodos para hacerlo más eficiente. Esto puede ser logrado como el paralelismo del flujo de una aplicación o como el paralelismo a nivel de red.
- *Services (Servicios)*: La aplicación entera se comporta como un servicio por demanda back-end (programa de respaldo, que efectúa las acciones de fondo, por ej.: DBMS).
- *Parallel Services (Servicios paralelos)*: La aplicación se ejecuta como un servicio múltiple, paralelizado, invocable, back-end.
- *(Programas paralelos fuertemente acoplados)*: Este tipo de ejecución es similar a la ejecución de servicios paralelos pero que están fuertemente relacionados en un proceso y comunicación inter-servicio, es decir, los servicios llaman a otros servicios en contraposición a servicios llamados sólo por los usuarios.

Se tiene la posibilidad de determinar qué grado de habilitación es apropiado para cada aplicación. Al seleccionar cualquiera de las anteriores, se deberá tener en cuenta en primer lugar las necesidades de los usuarios. Segundo, por lo que la aplicación hace, su funcionalidad, arquitectura y cómo fue escrita. Y por último por el tiempo que la aplicación necesita para correr en el grid.

Los seis grados de habilitación del grid descritos anteriormente no son mutuamente excluyentes ni tampoco niveles de habilitación. Son más bien estrategias, grados de adopción, o niveles de integración. Se interrelacionan

y pueden utilizarse como un medio de ayuda para atravesar las barreras, que van desde un grado de adopción relativamente simple como la ejecución de un trabajo batch, a un nivel más complejo de habilitación como por ejemplo la ejecución de servicios paralelos fuertemente acoplados.

Por ejemplo una aplicación existente podrá ser habilitada de manera tal que la mayoría de sus módulos computacionales intensivos en el uso de recursos, sean desplegados como un trabajo batch de instancia simple en el grid (1º grado de adopción). Adicionalmente, al modificar el mecanismo usado para poner en práctica la Infraestructura Grid (no a la aplicación misma), la aplicación misma puede “desplegarse” como si se tratase de un módulo habilitado en el grid ejecutándose como múltiples instancias independientes de un mismo trabajo batch (2º grado de adopción). Esto mejoraría el rendimiento global de la aplicación, comparado con la estrategia de tener una sola instancia corriendo en el grid.

Seguidamente la aplicación podría desplegarse de dos maneras:

- Se podría redefinir la interfaz grid una vez más y tener los módulos dispuestos en el grid ejecutándose como un trabajo batch subdividido (3º grado de adopción); resulta necesario hacer notar que lo que estará cambiando es la definición de una unidad de trabajo, no la aplicación. Una unidad de trabajo en este caso sería un arreglo de la unidad de trabajo original utilizada para implementar el primero y segundo grado de adopción.
- La aplicación puede ser re-diseñada de manera tal que los módulos disponibles en el grid ya no se ejecuten como una colección de trabajos batch sino más bien como servicios (4º grado de adopción).

Tomando cualquiera de los casos (3º o 4º) se provee la posibilidad de que la aplicación se despliegue y convierta en una colección de servicios paralelos (5º grado de adopción) y posteriormente en una colección de programas o servicios paralelos fuertemente acoplados (6º grado de adopción). Para llegar a este nivel, la aplicación necesita ser transformada de manera tal que el producto final no se parezca al original.

Hay factores a tener en cuenta con respecto a los seis grados de adopción y la manera en que cada uno se relaciona con los otros:

- Cada aplicación no necesita llegar al sexto grado de adopción. La mayoría no lo desea.
- Hay un valor considerable en todos los grados de adopción
- No se tiene que conseguir el objetivo en el primer intento.
- Para algunas aplicaciones, el esfuerzo de adoptar algunos grados de habilitación grid pueden consistir únicamente en el despliegue de tipos de trabajos mientras que la adopción de otros grados pueden requerir el cambio de la arquitectura y el código fuente de las aplicaciones.

Sin tener en cuenta el nivel de adopción, una aplicación de grid habilitada se ejecutaría en los nodos del grid como una aplicación regular. A causa de esto, puede notarse que no todas las aplicaciones necesitan tener en cuenta la existencia de la infraestructura grid, en cuyo caso, es posible que esa aplicación tenga que cambiar muy poco.

Considérese que, incluso sin tener en cuenta la infraestructura grid, una aplicación puede aprovecharla muy bien, como en el caso de los grids donde los nodos están escaneando máquinas de escritorio o servidores que solo se vuelven parte del grid al estar ociosas. Hoy en día, cualquier aplicación cliente que se ejecute como una aplicación grid habilitada utilizaría también la infraestructura grid. Cuanto más utilice la aplicación disponible en el grid a la infraestructura de dicho grid, mejor se ejecutará.

Este es el modelo básico para habilitación de un grid. Ya están definidas las seis estrategias denominadas grados de adopción para la habilitación de un grid e identificados los dos contextos en el cual se las puede aplicar (trabajos batch y servicios).

3.2.2 Software de Infraestructura Grid Orientado a Batch

Los primero tres grados de habilitación de grid son apropiados para la integración de código existente, utilizando software de infraestructura grid de tipo batch.

En términos sencillos esto puede ser definido como la disposición de una aplicación y un software de infraestructura unidos ambos por código adicional que permite el funcionamiento.

Hoy en día esta es la manera más fácil de habilitar las aplicaciones grid y en muchos casos no es necesaria la modificación del código fuente de la aplicación.

3.2.3 Software de Infraestructura Grid Basado en SOA

Los tres grados de adopción de grid de mayor nivel están bien preparados para ser usados en aplicaciones y software de infraestructura grid basado en SOA.

Cuando se habla de arquitectura orientada al servicio (SOA) se habla de servicios en general. Esto incluye servicios basados en OGSA y WSRF. Más específicamente, se habla del Globus Toolkit en sus diferentes presentaciones.

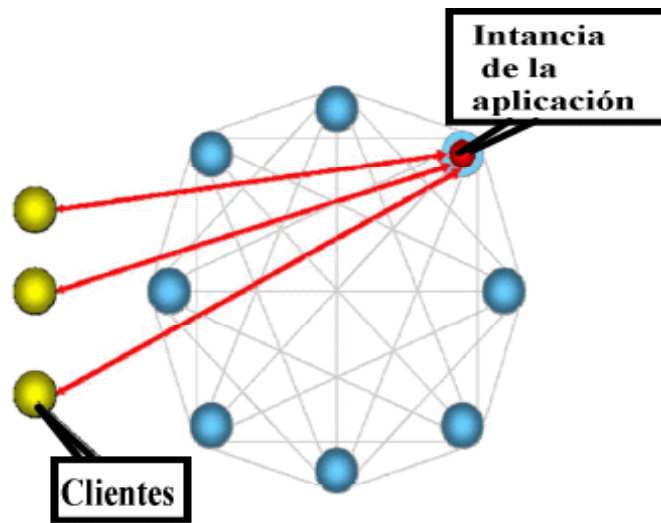
Se habla de *Servicios Web* como la base estratégica sobre la que se basan OGSA y WSRF.

La característica importante de estos grados de adopción de grid de niveles más altos es el hecho de que se puede necesitar modificar el código fuente del producto para que pueda correr como un servicio o como una colección de servicios. En el menor tiempo posible se tendrá que desplegar la aplicación como un *Servicio Web* o como una colección de *Servicios Web*. En el escenario del peor caso, se tendrán que reescribir la mayoría de los módulos del sistema, tal vez todos, para que se ejecute sobre el software de infraestructura grid basado en SOA.

Se necesita estar conciente, sin embargo, de que los productos muchas veces no están disponible, esto se debe principalmente al estado actual de los estándares. La reciente introducción de WSRF ha forzado a la comunidad de grid ha detenerse y re-evaluar algunos de los conceptos asumidos cuando el estándar era OGSI. Como resultado, es probable que algunos planes de habilitación de productos fueran re-evaluados y, como mínimo, es probable que todos los proyectos actuales de habilitación de productos, tuvieran que ser validados conforme a nuevos estándares.

Aunque es posible diseñar las aplicaciones basadas en OGSA y normas WSRF, se tendrán que hacer varias adopciones, para llenar los espacios de la especificación y se tendrá que re-arquitecturar (re-diseñar) el código basándose en las especificaciones anteriores.

Desde ahora en adelante se enfocará a los seis grados de adopción de grid en los dos contextos descritos anteriormente. En la próxima sección se comenzará con los tres grados más bajos de habilitación del grid.

Figura 3.5: 1^o Grado de Adopción.

3.3 1^o, 2^o y 3^o Grado de Adopcion

3.3.1 Batch Anywhere: 1^o Grado de Adopción.

Objetivo

En este nivel, el objetivo consiste en poder ejecutar una instancia del programa o aplicación en cualquiera de los nodos del grid. Dicho nodo será seleccionado por el software de infraestructura grid en tiempo de ejecución, y la selección del nodo se corresponderá con las características discretas de la carga de trabajo. Ver figura 3.5 de la página 77.

En este grado de adopción, el único aspecto de concurrencia consiste en que los nodos son concurrentes pero respecto a la ejecución de otros programas.

Características y Beneficios

Las características de este nivel de habilitación de grid son:

- Rendimiento, tiempos de respuesta y volumen aceptable de I / O (entrada / salida) para LAN y/o WAN.
- El aprovisionamiento no es caro. El mismo está basado en el licenciamiento.
- Seguridad adecuada.
- Tiempo aceptable de arranque.
- Puede requerir planificación por calendario, por trabajos previos, o por la llegada de archivos.

La ventaja principal que este grado de adopción es que proporciona independencia de localización por medio de la virtualización, es decir, una instancia simple de un trabajo ejecutándose en cualquiera de los nodos.

Propiedades del Ambiente

Un ambiente simple de ejecución de este grado de adopción exhibiría las siguientes propiedades:

- *Entrega de trabajos:* Las peticiones para ejecutar los trabajos son distribuidas en la infraestructura grid de manera estándar.
- *Seguridad:* Las demandas de trabajo, programas, archivos, etc., no se aceptan desde usuarios no autorizados. La identificación, autorización y autenticación son tratadas de una manera estándar. La información confidencial es encriptada por los nodos antes de la finalización de los trabajos.
- *Planificación del trabajo:* El planificador determina cuándo debería ser la ejecución basándose en políticas y prerrequisitos.
- *Ordenamiento del trabajo:* Los trabajos son priorizados de acuerdo a las políticas aplicadas.
- *Compartimiento de recursos:* Los recursos son compartidos entre todos los usuarios de acuerdo a las políticas aplicadas.

- *Asignación de trabajos:* Los trabajos son asignados a los nodos por un corredor de recursos basándose en políticas, requerimientos del trabajo, recursos de los nodos y contacto con la información.
- *Resguardo de recursos:* Los recursos subutilizados tales como servidores y estaciones de trabajos, son empleados.
- *Despliegue:* Los programas y archivos son movidos hacia donde se los necesitan.
- *Inicio del trabajo:* Los trabajos pueden ser iniciados por la infraestructura grid.
- *Monitoreo del trabajo:* Los trabajos que fallan pueden ser reiniciados en el mismo o en algún otro nodo.
- *Finalización del trabajo:* Un trabajo puede ser finalizado por la infraestructura grid si las políticas indican que otros trabajos deben tener prioridad.
- *Reinicio de un trabajo:* Si finalizó, un trabajo se reinicia cuando y donde sea apropiado.
- *Recopilación de resultados:* Los resultados son devueltos a los usuarios.
- *Acceso a la información:* Los programas puede acceder a la información libremente. La información puede estar protegida.
- *Concurrencia:* Otros tipos de trabajos se ejecutan al mismo tiempo.
- *Chargeback:* El uso es registrado y es cobrado a los usuarios.

Todas estas propiedades ayudan al entorno a ser más rápido y eficiente porque cuanto más rápido sean ejecutados los trabajos más rápido serán completados. Un ambiente con estas propiedades se comporta de una manera más flexible al asignar los trabajos.

Adicionalmente un ambiente que exhibe estas propiedades es más barato de operar.

Los trabajos tienen los costos operacionales más bajos cuando se ejecutan sobre nodos explotados y administrados eficientemente.

Finalmente, es posible afirmar que los entornos de este tipo tienen buena disponibilidad y flexibilidad. Con más máquinas capaces de ejecutar un trabajo, aun cuando alguna de ellas presente problemas, las otras pueden ocuparse de las tareas.

En resumen, se presentan beneficios considerables incluso en un ambiente donde la concurrencia se da solamente cuando una instancia de un trabajo puede ejecutarse en cualquiera de los varios equipos disponibles.

Licenciamiento

El programa debe ser usado legítimamente. Si el programa es licenciado o usa librerías licenciadas o middleware, entonces la licencia condiciona y determina cómo debe ser utilizado. A menos que, lo que se pretenda sea que el usuario licencie el software en cada nodo del grid, el programa no podrá usar más los dispositivos de protección de licencias o las licencias bloqueadas del nodo. En este nivel de habilitación del grid, se debe adoptar, como mínimo, un modelo de licenciamiento “*cualquier nodo de a uno por vez*”.

El modelo de licenciamiento estándar *X-Open Software License Use Management (XSLM)* de grupos abiertos incluye:

- *Elemento básico*: Empresa o sitio de licencias.
- *Nombramiento*: licencias basadas en usuarios registrados o en máquinas nombradas específicamente (también llamadas licencias bloqueadas de los nodos).
- *Concurrencia*: licencias restringidas para ser utilizadas por un número de usuarios concurrentes.
- *Cumulative o consumptive*: licencias basadas en el uso o tiempo de la ejecución de un trabajo. Estas podrían basarse en límites preestablecidos o en registros de auditoría de usos reales.
- *Capacidad*: licencias restringidas al uso en máquinas que tienen cierta capacidad informática inferior.

Este modelo puede ser adaptado de varias maneras. Puede ser manejado contractualmente en lugar de técnicamente, con el usuario responsable de

cumplir con las condiciones de la licencia. Puede también ser administrado vía referencia de auditoría, donde el programa o software de administración de trabajo mantiene un registro de uso o bien implementa un servicio de control de licencia por ejecución en la infraestructura grid, como por ejemplo el *Global License Broker del Platform*.

También se podría administrar el licenciamiento mediante un software de control de licencias de usos de programas por ejecución como el *Isogon LicensePower/iFOR*, *Macrovision FLEXnet*, o el *IBM License Usage Manager*.

Seguridad

La seguridad debe ser administrada, idealmente, fuera del programa. La responsabilidad de determinar quién puede ejecutar un programa debería ser parte del ambiente. El programa no debería ejecutarse con el ID de un usuario no autorizado.

Muchos programas crean archivos intermedios con datos confidenciales y esos archivos no pueden ser eliminados si el programa falla. En estos casos, se debe al menos documentar, en qué instrucciones de operaciones ocurren.

La Afinidad del Nodo

No se debe asumir que la ejecución del programa se realiza en el mismo nodo que en la ejecución previa o próxima. Se debe documentar qué archivos se necesitan que estén disponibles, cuáles se crean y cuáles cambian. Esto permitirá a las personas que despliegan programas planificar cualquier necesidad de aprovisionamiento y además considerar tecnologías de caching, los ejemplos más concretos son el IBM's SAN File System o Avaki Data Grid.

Administración del Trabajo

Se debe asumir que el programa sería ejecutado por el sistema de administración de trabajo, no por un humano mediante un comando de prompt o sistema de menú. Los estándares de entrada/salida como **stdin**, **stdout**, y **stderr** deben ser redireccionados hacia los archivos.

Tráfico de Red

Traffic LAN en un Solo Sitio Si la aplicación se va a ejecutar en cualquiera de los nodos dentro de un sitio utilizando solo recursos intra-sitio, entonces se debería considerar el impacto del tráfico de la LAN en las recomendaciones.

El tráfico de la LAN comienza desde el aprovisionamiento del programa y los archivos de prerequisites para el nodo antes de la ejecución hasta el traslado de los resultados creados localmente en alguna otra parte.

El tráfico adicional parte desde el uso de bases de datos fuera del nodo, mensajería, sistemas de transacción y otros recursos. Se debería considerar la ejecución de otros programas en el mismo u otros nodos, que también estarían usando la LAN.

Tráfico WAN en Múltiples Sitios En el caso de que la aplicación se ejecute en cualquiera de los nodos dispuestos en más de un sitio, el tráfico WAN tiene un impacto en el rendimiento y la confiabilidad.

Por ejemplo, los archivos que no fueron “cacheados” (copiados a caché) de cada sitio se volverán parte del proceso de aprovisionamiento. Además antes de la finalización del programa los resultados necesitarían ser enviados de regreso a su último destinatario. El acceso remoto a contenido oculto, vía NSF u otras técnicas, sería viable sólo si la cantidad de accesos y la totalidad de datos transferidos no supera el tiempo de ejecución del programa por un inaceptable periodo de tiempo.

El Uso de las Bases de Datos en los Despliegues Multi-sitio Aparte del tráfico WAN, hay consideraciones adicionales con respecto al acceso a las bases de datos u otros recursos transaccionales en una o más localizaciones remotas. Cuando un DBMS tiene múltiples clientes, es esencialmente importante, minimizar el tiempo en que estos se bloquean.

Debido a que la duración del bloqueo es corta y el control de seguridad muy bueno, el poseedor de los datos puede permitir a un programa realizar las actualizaciones en un procedimiento almacenado que no sería permitido en un programa del lado del cliente.

Las tecnologías de base de datos federadas como el *DB2 Information Integrator* pueden ayudar a reducir el tráfico WAN y el código del lado del cliente

cuando los datos son unidos desde múltiples localizaciones.

Otros Problemas

En una situación ideal, se podría desplegar simplemente una aplicación sobre un ambiente grid sin tener que cambiar nada. Pero se sabe que éste, no es el caso.

La ejecución en un ambiente grid propone desafíos, que no existen en otros ambientes distribuidos. Algunos de estos desafíos deben ser establecidos teniendo en cuenta los acuerdos de nivel de servicios y seguir un plan de funcionalidad grid.

La siguiente, es una lista de cambios que se deben considerar para ser implementados sobre una aplicación al ejecutarla como un trabajo batch de instancia simple en un grid. Esta lista y los cambios sugeridos no son obligatorios, pese a ello, es bueno tenerlos en cuenta:

- Ampliar lo que pueda ser controlado por parámetros de programa, por ejemplo, memoria máxima o número de procesos a explotar.
- Permitir al programa ser cancelado y luego reejecutado sin causar problemas. Posiblemente sea más tarde y en otra máquina.
- Permitir al programa realizar el check pointing ocasional, de tal manera que cuando éste se reejecute después de una cancelación no necesite hacer todo el trabajo nuevamente. Proporcionar los parámetros checkpoint-andrestart para que el usuario pueda especificar con qué frecuencia los registros checkpoint son grabados. Por ejemplo cada n registros de entrada, cada m minutos, etc.
- En ausencia de archivos checkpoint, permitir que el programa sea reejecutado desde el comienzo.
- Mantener información de consumo de recursos para usar durante la planificación de despliegue, por ejemplo la memoria, modelos de e/s, registros de tiempos y usos de CPU por salidas y por unidades de trabajo. Se deben registrar los requisitos en un estándar de la industria bien documentado de manera que puedan ser entendidos por personas encargadas del despliegue de la aplicación.

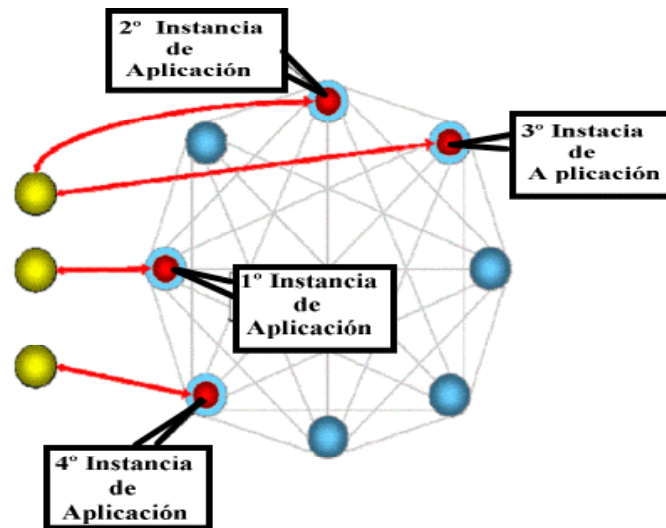


Figura 3.6: 2º Grado de Habilitación Grid.

3.3.2 Independent Concurrent Batch

Objetivo

La idea principal en este grado de adopción consiste en que varias instancias de una misma aplicación se ejecuten concurrentemente. Por ejemplo, Jhon ejecuta sus datos en una aplicación llamada Análisis-A, mientras que Jane ejecuta sus datos a través de otra instancia de la aplicación Análisis-A. Este no es el mejor avance desde el grado anterior, pero tiene ciertas consideraciones adicionales.

El modelo de ejecución para este grado de adopción es ilustrado en La figura 3.6 de la página 84.

Características y Beneficios

Se puede decir, que este grado de adopción es una extensión del primero. Esto es cierto en el sentido de que la mayoría de las características del grado anterior se aplican a éste. No obstante, en adición a aquéllas, este grado de habilitación de grid, en particular tiene las siguientes características:

- Varias instancias ejecutándose independientemente sin interferencias.
- Existencia de concurrencia. Esto significa que un trabajo **x** (aplicación) ejecutado en **a** puede ejecutarse concurrentemente con el trabajo **x** (aplicación) ejecutado en **b**.
- Las Bases de Datos no exhiben puntos de saturación o bloqueos.

La estrategia de despliegue es la misma que la mostrada en la figura 3.5 de la página 77. Como así también las consideraciones para la ejecución de aplicaciones y determinación de las propiedades del ambiente.

Hay ciertas diferencias, como por ejemplo, en los casos del licenciamiento y los requerimientos para evitar la interferencia entre las instancias. Todas las otras consideraciones tomadas en el grado anterior, se aplican también en este grado de adopción.

Problemas de Licencias

Si una aplicación es licenciada, el modelo de licencia y la tecnología usada deben forzar al modelo de licenciamiento para que permita la concurrencia. Además de las técnicas descritas en la sección anterior, las licencias basadas en la concurrencia pueden ser una opción, fundamentándose en la concurrencia de máquinas, procesadores o instancias.

Los Problemas de No Interferencia

Ahora que las múltiples instancias pueden ser ejecutadas concurrentemente, es muy importante que éstas no interfieran una con la otra.

La aplicación debe brindar la posibilidad de que la primera instancia iniciada no sea la primera en finalizar. Esto significa que el rendimiento o las actualizaciones de una instancia no deben recubrir a otra. Además, la aplicación no debe crear patrones de bloqueos al usar recursos transaccionales.

Es poco probable la presencia de problema de alta actividad. En el diseño de las bases de datos donde las instancias de los programas actualizan filas comunes, presentarían inconvenientes. Si ese es el caso, se debería encontrar la manera de modificar la aplicación para que cada instancia actualice filas

no comunes o bien actualicen las filas compartidas con menos frecuencia y sin permanecer bloqueado por una cierta cantidad de tiempo.

Considerar la validez y habilidad del programa para reiniciar si una instancia concurrente comparte un suministrador de recursos de incremento común (por ejemplo un número serial) o un generador de número aleatorio, etc.

3.3.3 Parallel Batch

Objetivo

El objetivo principal consiste en tomar el trabajo batch de cada usuario, subdividirlo y distribuirlo por los múltiples nodos, y por último recolectar y luego agregar los resultados.

Es el momento de comenzar a usar los conceptos de clientes y servicios, que realmente no son aplicados en los dos primeros grados de adopción grid. En esos dos escenarios la entidad más cercana al cliente es un nodo peticionador de subdivisiones de trabajos, cuyo fin es el de someter los trabajos a un planificador. Funciona de manera tal que, el planificador efectúa la entrega de los trabajos al software de infraestructura grid para su ejecución en los nodos disponibles.

En el caso de los trabajos batch paralelos, el cliente es el peticionador que realiza la subdivisión y la entrega de los trabajos para su ejecución, para más tarde realizar la agregación de los resultados. Este modelo de ejecución es ilustrado en la figura 3.7 de la página 87.

Este grado de adopción es útil cuando el problema puede ser subdividido en problemas más pequeños.

En adición a las características exhibidas para los dos primeros grados de adopción, que también son aplicadas a este grado de adopción en particular, se tiene las siguientes características:

- Paralelismo, es decir, la habilidad de subdividir y distribuir el trabajo entre los nodos.
- Multiplicidad o habilidad para dividir la funcionalidad de la aplicación en una relación uno a muchos entre los clientes y los trabajos del servidor.

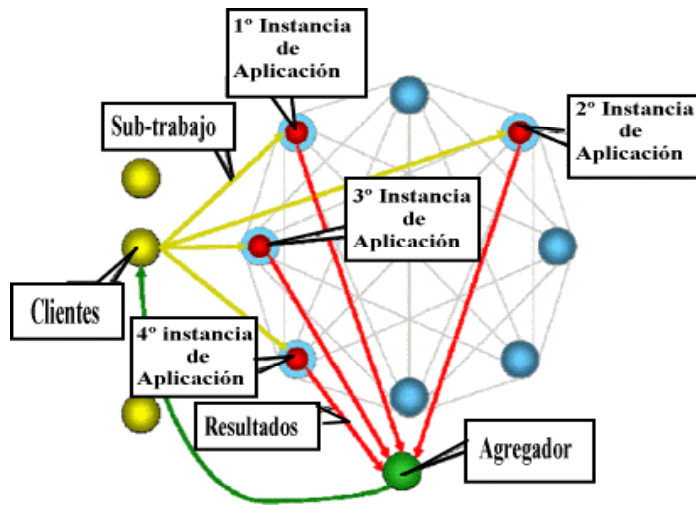


Figura 3.7: 3^o Grado de adopción del Grid.

- Los clientes pueden separar el trabajo en trabajos más pequeños del servidor para la difusión.
- Los trabajos del servidor se comportan como las instancias múltiples e independientes justamente como en el grado de habilitación anterior.
- Después de recoger los resultados, el cliente puede además agruparlos.

Entre los beneficios de la aplicación de este grado de habilitación debemos mencionar los siguientes:

- Nacimiento de un nuevo modelo de programación conocido como Paralelismo Paramétrico donde cada trabajo se ejecuta en algún programa, pero con diferentes parámetros o entradas.
- Tiempos de completitud relativamente buenos.
- Tiempos de recuperación también buenos. Si cualquier sub-trabajo falla sólo será necesario reejecutarlo en lugar de reiniciar el trabajo entero.
- Habilidad para salvar los ciclos ociosos desde los nodos del grid.

- Granulabilidad y efectividad buena en los costos, que permite el uso de máquinas baratas en un número amplio para trabajar, en lugar de utilizar servidores costosos y complejos.

Subdivisión, Procesamiento y Agregación en los Trabajos

Para que este grado de adopción funcione, el programa cliente debe comportarse como una unidad de entrega de trabajo. Esto significa que se debe poder dividir el trabajo para subsecuentemente distribuirlo por el resto. Ese resto sería el software de infraestructura grid.

Para la mayoría de las aplicaciones el programa cliente probablemente no exista y el grado de dificultad asociado para su escritura dependerá de la arquitectura de la aplicación, es decir, qué es lo que necesita ser subdividido y cómo se define la unidad de trabajo. La agregación de resultados será tan fácil o tan difícil como la arquitectura de los permisos de la aplicación.

Se necesita estar conciente de lo que este grado de habilitación de grid requiere de la aplicación. Si se está dispuesto a gastar tiempo y dinero en la modificación de la aplicación, se debería al menos ajustar los parámetros y los componentes del servidor secundario, de tal manera que cada ejecución pueda ser parte del trabajo.

También puede que se tenga que optimizar los algoritmos a ser usados en el grid. Por ejemplo, se puede inclinarse por algoritmos que toman mucho más tiempo utilizando una sola máquina, pero avanzará favorablemente cuando varias máquinas sean utilizadas. Hay varias técnicas disponibles para optimizar algoritmos en ambientes distribuidos amplios.

Más Acerca del Programa Cliente

El programa cliente acerca del que se ha hablado no tiene que ser muy complejo. Pero tampoco muy simple.

Se podría implementar una unidad de entrega de trabajo como un conjunto de scripts de la shell. Estos scripts pueden realizar las tareas discretas tales como dividir la totalidad del trabajo y entregar cada una de las subunidades de trabajos para su ejecución en la infraestructura grid (esta es la manera de tener más subunidades que nodos obreros en el grid) y notificar además a la

infraestructura grid que las subunidades están, y debería ejecutar la aplicación del servidor principal una vez por cada subunidad.

Si la agregación es requerida, el mismo esquema de scripts que ejecuta el cliente puede también invocar a la infraestructura grid para recoger los resultados o invocar otra categoría de programa de servidor (funciones de negocios) que agregará los resultados desde las subunidades.

El Software de Infraestructura Grid

La infraestructura grid debería expandir las subunidades de trabajo por los nodos para su procesamiento; afirmando de que todas las subunidades se ejecutan correctamente; y facilitando la agregación de resultados. La misma debería devolver los resultados a algún punto central para la agregación o por lo menos permitir la accesibilidad de los mismos.

Permanecer Independiente de la Infraestructura Grid

Evitar el uso del software de infraestructura grid en la aplicación del lado del servidor siempre que sea posible.

En general, es suficiente para el software de infraestructura grid ejecutar la aplicación del servidor. La infraestructura grid esta encargada de ejercer la seguridad, la autoridad, la accesibilidad de recursos, las variables de ambiente, etc. Y una vez que esté todo seguro, la aplicación se ejecute sola.

Algunas aplicaciones pueden tener gran necesidad de implementar el software de infraestructura grid. Pero para mantener cierto grado de independencia, los módulos que interactúan con la infraestructura grid deberían ser implementadas como componentes sustituibles (un nuevo rostro en la terminología de modelos de diseño) siempre que sea posible. De nuevo la modularidad y granularidad permiten la creación del grid de manera más fácil.

Acerca de los Componentes del lado del Servidor

Variable Mid-grain Al ajustar el programa del servidor, se le debería permitir al cliente especificar el número de ítems de trabajos a ser procesados en cada ejecución.

Este es el método “mid-grain”, una práctica no obvia muy buena. Desafortunadamente, el método obvio fine-grained, donde una ejecución del servidor es una unidad de trabajo, ocasiona varias ineficiencias. Entre otras cosas las más destacable son que, el cliente necesita separar el trabajo en ítems individuales, la infraestructura grid deberá priorizar, planificar y arrancar el programa del servidor una vez por cada ítems y además se deberá programar el inicio y finalización de las operaciones de gestión interna por cada ítems individual. Lo cual no es conveniente.

Al permitir al cliente especificar cuántos de los ítems incluir en cada subunidad, el número y el tamaño de las subunidades pueden ser optimizados por el equipo de administración. Estos pueden efectuar los ajustes basándose en las características del trabajo, la infraestructura grid, el sistema en el grid, y la variabilidad en el tamaño de las unidades de trabajo.

Un ítem de trabajo podría ser parte del procesamiento en un cálculo, donde algunos de esos ítems tienen más actividades que otros durante el mes. Por ejemplo en un grid de nodos unequal-powered, con cargas moderadas, pero de tiempos variables de ejecución procesados en orden de tamaño decreciente, permitirá asegurar que casi todos los nodos sean explotados hasta que el trabajo finalice.

La No Interferencia e Independencia

Se trata el problema de la interferencia entre las ejecuciones independientes en representación de diferentes usuarios en la descripción del segundo grado de adopción.

Los mismos problemas son aplicados en este grado dado que los múltiples sub-trabajos se ejecutan concurrentemente en representación de un solo usuario, con la condición de que los sub-trabajos no interfieran unos con otros.

Como en el caso anterior, la aplicación debe impedir la existencia de modelos de bloqueo al usar recursos transaccionales. Debe evitar además los hotspots de middleware, y los de bases de datos. Se debería actualizar las filas compartidas de una forma no frecuente e incluso sin la advertencia de bloqueos por una cierta cantidad de tiempo.

Licenciamiento

Desde la perspectiva del licenciamiento, el logro de la concurrencia estuvo a cargo del segundo grado de adopción del grid. Sin embargo, en este caso, el software de control de licencia puede que necesite manejar una nueva situación. Por ejemplo, si la aplicación es licenciada por usuario, en lugar de licenciar por máquina-usuario, un usuario explotando dos máquina y otro usuario explotando cuatro, requerirá de dos licencias no de seis.

El Agregador

A no ser que agregar los resultados sea tan simple como encadenar archivos de salida en cualquier orden, la aplicación debe incluir un programa separado que agregaría los resultados. Se debe considerar que no es trabajo del programa de agregación recoger los resultados en un lugar. Esta responsabilidad debería encargarse a la infraestructura grid.

Si la agregación proviene desde un contenido localizado en una dirección que debe ser pasada como un parámetro (una URL, o una ruta) entonces el agregador (programa) no tiene que interactuar recíprocamente con la infraestructura grid para conseguir esa información. En este caso, el contenido podría especificarse como un parámetro de entrada al agregador y con eso sería suficiente.

Si el agregador debe utilizar la infraestructura de grid, debería dejar de lado el código de productos específicos que usan las técnicas descritas en la sección anterior “permanecer independiente de la infraestructura”.

Anteriormente fueron mencionados los tres primeros grados de adopción para la habilitación de grid, preparados para habilitar aplicaciones existentes en los productos de infraestructura grid orientados a batch (figura 7).

En la próxima sección se mostrará la incorporación de algunas modificaciones a estos tres grados de habilitación de grid.

Primero, introduciremos el modelo arquitectónico básico que se ajusta a la mayoría de los casos al habilitar código existente para ejecutar un software de infraestructura grid orientado a batch. Después se discutirán algunas estrategias de habilitación basadas en las arquitecturas más comunes que se han encontrado durante los últimos años. Se introducirán dos escenarios básicos

para aplicaciones distribuidas de plataforma específica y se estudiará las variaciones de la mayoría de las arquitecturas comunes de cada escenario y las maneras más comunes de crear esas arquitecturas apropiadas en un modelo de habilitación.

3.4 Habilitación de Código Existente

3.4.1 Existe un Modelo

La mayoría de las técnicas o estrategias de integración usadas para habilitar código existente empleando un software de infraestructura grid orientado a batch son similares. Esto significa que existe un modelo a seguir. Este modelo es el que se puede usar para lograr los tres grados de adopción más bajos de grid (1º, 2º y 3º grado de adopción).

Ya se ha mencionado que, el escenario básico para los tres grados de adopción más bajos de grid consisten en un programa que toma parámetros de línea de comando y utiliza archivos y/o bases de datos como los especificados en los parámetros de línea de comando. También se mencionó que el programa puede ser licenciado, en caso de que la infraestructura grid requiera capacidades de administración de licencia.

En general, habilitar una aplicación existente en los tres primeros grados de adopción (1º, 2º y 3º) para correr en un grid requiere que el usuario envíe peticiones a una aplicación cliente, que actúa como un peticionador para la infraestructura grid. El cliente para la aplicación cliente puede ser el usuario actual o un portal.

La infraestructura grid soporta el despliegue de la aplicación actual, la que se convierte en proveedor para la infraestructura grid misma. Esto se ilustra en la figura 3.8 de la página 93.

La clave está en que el programa cliente (unidad de entrega de trabajo) debería comunicarse con la infraestructura grid como si se estuviera comunicando directamente con la aplicación. La manera más simple de lograr esto es tener un programa cliente emitiendo instrucciones de tipo línea de comando a la aplicación virtualizada.

Implementar este escenario resulta fácil cuando la aplicación es un programa autónomo con requerimientos mínimos de despliegue. Pero, cuando se

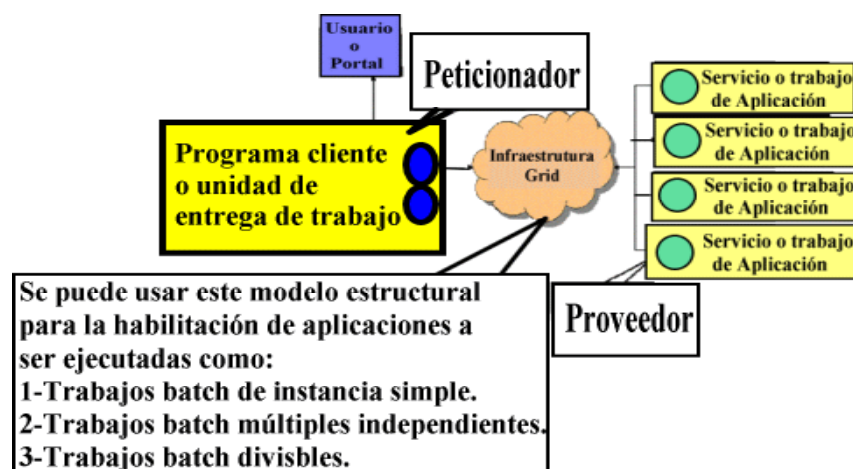


Figura 3.8: Modelo de Integración Para Habilitar Código Existente.

trata de aplicaciones integradas el proyecto entero puede requerir un poco más de creatividad.

3.4.2 Los Escenarios Conocidos

La habilitación de código existente usando un software de infraestructura grid orientado a batch involucra un número finito de escenarios conocidos. Hay dos escenarios, y ambos involucran esencialmente al mismo tipo de aplicación:

- Habilidad de *Aplicaciones Distribuidas de Plataforma Específica*, que incluye aplicaciones cliente/servidor, transaccionales y orientadas a batch escritas antes de la aparición de las aplicaciones Web.
- Habilidad de *Aplicaciones Disponibles Para la Web*, que incluye aplicaciones distribuidas de plataforma específica a las que se les dio una “interfaz” o “envoltura” Web para hacerlas trabajar como aplicaciones Web. En muchos casos esta estrategia de integración implica la habilitación de aplicaciones *Servlet-Centric* o *Database-Centric*.

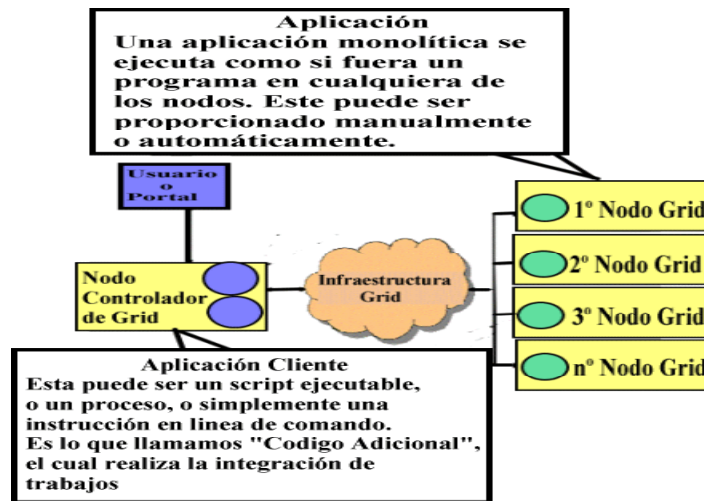


Figura 3.9: Despliegue de Aplicaciones Monolíticas Usando el Modelo de Habilitación Estándar.

Habilitación de Aplicaciones Distribuidas de Plataforma Especifica

Como se mencionó anteriormente este escenario se aplica a aplicaciones cliente/servidor, transaccionales y orientadas a batch. Hay dos tendencias estructurales que prevalecen en estos tres tipos de aplicaciones: *Monolíticas* y *Modulares*.

El Caso de las Aplicaciones Monolíticas En general, desplegar aplicaciones *Monolíticas* de plataforma específica en un Software de Infraestructura Grid Orientado a Batch es tan simple como instalar la aplicación sobre todos los nodos del Grid y escribir “código adicional” que integre las peticiones del usuario, efectúe la entrega de parámetros, y realice las llamadas a los programas desde la infraestructura grid. Esto es ilustrado en la figura 3.9 de la página 94.

El “código adicional” mencionado anteriormente es el que hace de éste todo un trabajo de integración. En muchos casos, se puede integrar aplicaciones *Monolíticas* y un producto de infraestructura grid orientado a batch a través de guiones o scripts. Se puede utilizar Perl, Python, u comúnmente script de la shell para integrar las peticiones de los usuarios, la entrega de los parámetros

y llamadas a aplicaciones dentro del contexto del software de infraestructura grid.

Riesgo Existente Siempre hay un riesgo y en este caso, tiene que ver con lo que la aplicación *Monolítica* hace y cómo lo hace.

Las aplicaciones *Monolíticas* tienden a intentar todo y para todos. Esta es la razón por la que son *Monolíticas* (los módulos no son confiados sólo a ciertas personas, sino que a todos en general). Algunas veces, las aplicaciones *Monolíticas* tienen integradas entre otras cosas funcionalidades grid. Esto, y la manera en que tales funcionalidades pueden ser implementadas, determinarían si la aplicación puede ser ejecutada en un grid.

Por ejemplo, una aplicación *Monolítica* que no tiene ninguna funcionalidad de la infraestructura grid incorporada sería más fácil de habilitar que una aplicación similar que, además de hacer lo que se supone que hace, controla tareas como determinar qué peticiones, qué instancia procesa, o qué tablas de las bases de datos necesitan estar bloqueadas en nombre de un usuario dado, o bien cuándo la afinidad transaccional necesita ser reforzada.

Si la funcionalidad grid incorporada puede ser “suspendida” desde el interior de la aplicación, entonces esos módulos de código monolíticos serían capaces de ejecutarse sin ningún problema sobre un software de infraestructura grid orientado a batch. Por otro lado, si la funcionalidad grid se establece interiormente en la aplicación y no puede ser suspendida, entonces se está en presencia de un problema.

En general, la magnitud del trabajo necesario para “suspender” la funcionalidad incorporada de la infraestructura grid es muy grande.

El Caso de las Aplicaciones Modulares En general, habilitar aplicaciones *Modulares Distribuidas de Plataforma Específica* será más fácil que habilitar aplicaciones *Monolíticas*. El motivo por el cual se asegura esto, es que las aplicaciones *Modulares* brindan opciones sobre cómo desplegarlas. Hay ciertos riesgos como los expuestos en los casos previos pero, en este grado, puede haber maneras más fáciles de llegar a la habilitación.

Suspensión de los Módulos Una de las mayores ventajas de las aplicaciones *Modulares* está en la posibilidad de suspender los módulos cuando

sea necesario. De esta manera, cualquier funcionalidad relacionada con el ambiente puede ser dada directamente a la infraestructura grid.

Como en el caso de las aplicaciones *Monolíticas*, la existencia de características grid incorporadas y si éstas pueden ser suspendidas o sustraídas, también determinarían el grado de dificultad para la habilitación del grid.

La diferencia esta en que la mayoría de las aplicaciones *Modulares*, cuando tienen cualquier funcionalidad grid incorporada, esa funcionalidad se concentraría en un único módulo o un grupo especializado de módulos. Esto en teoría debería hacer más fácil el trabajo de suspender o eliminar los módulos en su totalidad.

Riesgo Existente Hay un riesgo: la comunicación ínter-módulo. El grado de dificultad en la suspensión, o en la extracción de los módulos de la aplicación dependerá de cómo los diseñadores hayan **implementado la comunicación ínter-módulo**.

Por ejemplo es común en las aplicaciones de este tipo tener un módulo especializado ocupándose de todas las llamadas a las bases de datos. En algunos casos, el módulo no solamente actúa como cliente universal de base de datos soportando drivers ODBC o JDBC , sino que también hace algo que puede ser llamado como (Planificación de Acceso a Tabla), que es una especie de mecanismo intra-aplicación de tabla-bloqueada que permite a la aplicación manipular el bloqueo de las tablas independientemente de la base de datos.

Tener un cliente universal de la base de datos es una buena idea. Sin embargo, si la aplicación está destinada a ser habilitada en un grid, resultará favorable dejar el bloqueo de las tablas para la infraestructura grid. Así, todo lo que se necesita es sustituir el módulo para un cliente regular de la base de datos, desplegar el en el grid de datos y lo que se obtiene finalmente es una aplicación grid habilitada.

Hay otro aspecto de la comunicación Inter-modulo. Si la aplicación va a ser desplegada en un grid computacional puede haber varias instancias de varios módulos ejecutándose concurrentemente en el grid. Si el software de infraestructura grid no puede relevar el mecanismo de transporte, o si el mecanismo de transporte mismo no funciona en un ambiente grid, la aplicación simplemente no trabajaría como se esperaba. Entonces el grado de dificultad para el proyecto de habilitación de grid sería directamente proporcional al esfuerzo de reemplazar el mecanismo de comunicación inter-módulo.

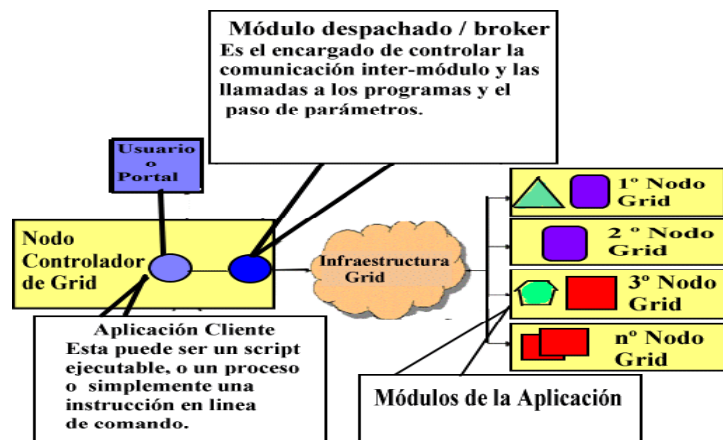


Figura 3.10: El Despliegue Ideal Para las Aplicaciones Modulares.

Estrategias de Despliegue Para Aplicaciones Modulares “Escenario del mejor caso”

Una aplicación modular ideal manejaría la comunicación inter-módulo vía módulo despachador o broker. Esto permitiría a los módulos ser desplegados en cualquier parte del grid. La comunicación inter-módulo siempre ocurriría a través de un broker. Esto es ilustrado en la figura 3.10 de la página 97.

Hay dos comportamientos deseables para este escenario.

Primero, los módulos de aplicación deberían ser atómicos como para permitir que sean desplegados independientemente entre sí. El módulo despachador / broker debería controlar la comunicación inter-módulo y el intercambio de datos. En cuanto a las bibliotecas compartidas, la infraestructura grid debería estar habilitada para administrarlas si es que fueron instaladas como parte de la instalación del sistema completo. Si no, podrán ser incluidas como parte de las políticas de aprovisionamiento distribuidas por todos los módulos para que todos ellos tengan una copia local.

Segundo, los módulos de la aplicación deben ser granulares tanto como para permitir que las instancias múltiples de un mismo módulo se ejecuten concurrentemente, por lo menos, en la misma máquina. Esto significa que un módulo debería preocuparse por sus propios resultados de agregación. La agregación de resultados a nivel de aplicación podrían ser efectuada por el

módulo despachador / broker o a través de las base de datos.

Este escenario ofrece un alto grado de flexibilidad para afinar el rendimiento del ambiente grid habilitado. Se pueden manipular entradas y salidas en el nodo controlador de grid y en el agregador de resultados y mostrarlos finalmente por las diferentes combinaciones de módulos en los diferentes nodos.

Por ejemplo, si en un momento dado se verifica que el módulo con el cuadrado rojo (ver nuevamente la figura 3.10 de la página 97) es computacionalmente uno de los más intensivos y se percibe que agregándole más CPU la aplicación finaliza el trabajo en menos tiempo, se podrá proporcionar a ese módulo una máquina completa, posiblemente una con más CPUs en su infraestructura y después realizando una supervisión y medición el sistema durante cierto tiempo, se podrá verificar que hay suficiente lugar en otra máquina como para correr no sólo el módulo Green Pentagon (figura 3.10 de la página 97) sino que también seguramente una instancia del módulo Red Square (figura 3.10 de la página 97).

Si las condiciones de trabajo cambian, todo lo que necesita hacer es monitorear el sistema nuevamente y reconfigurar la tipología de aplicación.

En un futuro no muy lejano, ya no se tendrá que hacer lo mencionado anteriormente cada vez que las condiciones de trabajo cambian. Todo lo que se tendrá que hacer es definir un perfil de calidad de servicio para la aplicación y un acuerdo de nivel de servicios (SLA) para los usuarios en el nivel de Orchestrator y las características autonómicas del grid permitirán la monitorización del sistema, proporcionando nuevos nodos a el módulo cuadrado rojo (o cualquier otro módulo) y reconfigurando la topología de aplicación cuando sea necesario para permanecer dentro de los parámetros SLA por los que se habrá pagado.

“El escenario más común”

Desafortunadamente, la mayoría de las aplicaciones no reaccionan de la mejor manera. En algunos casos, el encapsulamiento del módulo no es lo suficientemente atómico como para permitir un despliegue verdaderamente independiente. En otros casos, el módulo despachador / broker no controla completamente toda la comunicación inter-módulo y el intercambio de datos pero, sin embargo, algunos módulos llaman a otros directamente, obligándolos a residir en la misma máquina.

La agregación de resultados también puede representar un problema, es-

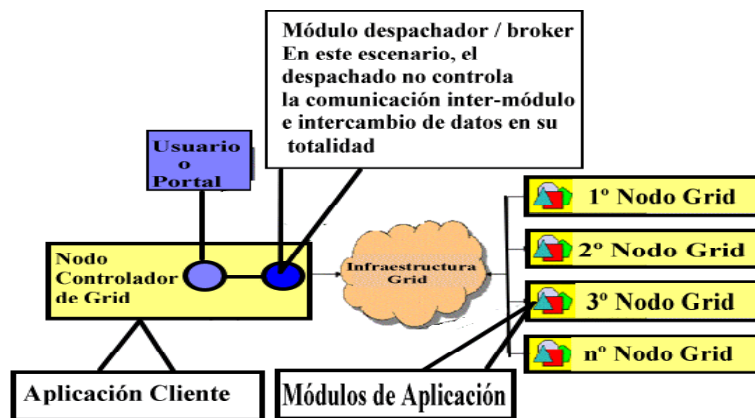


Figura 3.11: El Escenario Más Común Para el Despliegue de Aplicaciones Modulares.

pecialmente cuando el despachador / broker no tenga la capacidad de administrar la comunicación inter-módulo totalmente. Algunos módulos pueden alimentar sus resultados en otros módulos en lugar de pasarlos previamente al despachador.

En cualquier situación, el escenario más común es desplegar la aplicación entera sobre todos los nodos del grid, como se muestra en la figura 3.11 de la página 99.

En el mismo escenario, se podría imaginar una aplicación *Monolítica* como una aplicación *Modular* de módulo-simple y tratarla como tal al diseñar una estrategia para la administración de agregación de resultados en el caso de múltiples instancias concurrentes.

Habilitación de Aplicaciones Disponibles Para la Web

Una *Aplicación Disponible Para la Web* no es una verdadera aplicación J2EE . Se llaman *Aplicaciones Disponibles Para la Web* a las que fueron escritas originalmente como *Aplicaciones Distribuidas de Plataforma Específica* o como aplicaciones basadas en Java J2EE pero ejecutadas como aplicaciones web gracias a una apariencia web final.

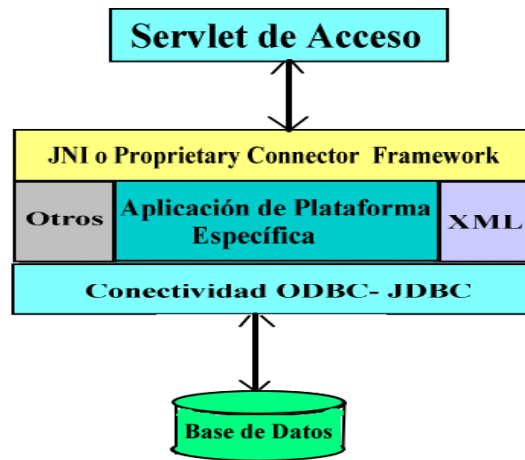


Figura 3.12: El Modelo Estructural Servlet-centric más Común.

Las estructuras más comunes son conocidas como *Servlet-centric* y *Database-centric* y cada una propone sus propios desafíos en la habilitación de un grid.

Aplicaciones Servlet-centric Las aplicaciones *Servlet-centric*, en general, siguen el modelo estructural ilustrado en la figura 3.12 de la página 100.

Lo que se tiene en la figura anterior es una aplicación de plataforma específica que, en algunos casos, se modifica para soportar tecnologías tales como XML y otras, y se habilita para comunicarse con una Máquina Virtual Java vía JNI o estructura del conector principal. En cuanto al soporte de las bases de datos, lo común es usar puentes ODBC - JDBC o simplemente utilizar ODBC.

Cuando se “migran” para ejecutar en servidores la aplicación J2EE, las aplicaciones Servlet-centric interactúan con los clientes vía un Servlet de acceso (gateway) que lanza las peticiones al conector así como a la aplicación actual. Una “migración” típica a un servidor de aplicaciones tal como el WebSphere Application Server : Servidor de Aplicaciones WebSphere (WAS) se ve en la figura 3.13 de la página 101.

Habilitar una aplicación que siga este modelo de ejecución en un grid requerirá al menos del cumplimiento de los siguientes pasos:

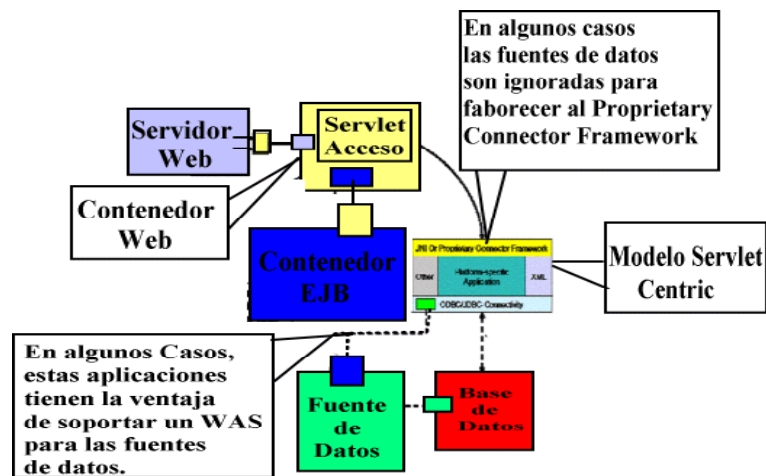


Figura 3.13: Estrategia Típica de Habilitación Web Servlet-centric.

- Se modifica el descriptor de despliegue para que sólo un componente sea actualmente desplegado sobre WAS (*Servidor de Aplicación WebSphere*), es el Servlet de acceso el que se convertirá en interfaz para los usuarios.
- Tomar el módulo java del entorno del conector (JNI o proprietary) que actúa como interfaz para la aplicación central y despléguelo como una aplicación java autónoma. Este será el programa cliente o la unidad de entrega del trabajo.
- Desligue el centro de la aplicación como una aplicación *monolítica* o como una aplicación *modular* la que a su vez aplicará sobre una infraestructura grid orientada a batch.

El escenario resultante es ilustrado en la figura 3.14 de la página 102.

Podría ser necesario cambiar algunos de los supuestos originales al hacer esto.

Se debe tener en cuenta que ésta es sólo una manera simple de hacerlo. Puede haber buenas maneras de estructurar el modelo de despliegue dependiendo de las características de la aplicación. La mejor solución debería proporcionar los mejores resultados en términos de viabilidad, esfuerzo, utilidad, y administración.

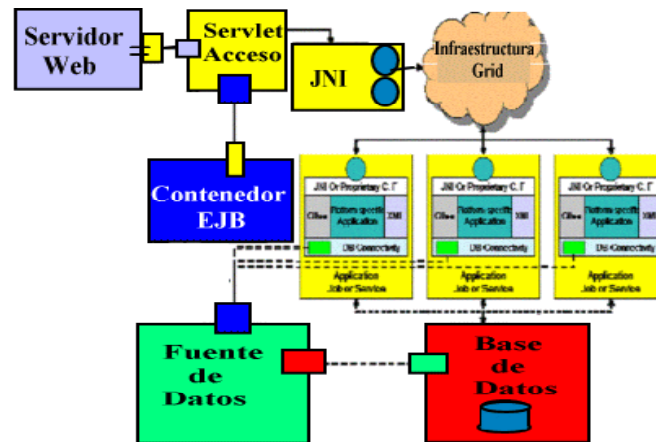


Figura 3.14: Despliegue grid típico de Servlet central.

Riesgos Los problemas que afectan a las aplicaciones *monolíticas* y *modulares* pueden también afectar a las habilitaciones grid *Servlet-centric*. Aparte de estos problemas, también pueden presentarse aquellos relacionados con el rendimiento de la aplicación.

Las aplicaciones *Servlet-centric*, en la mayoría de los casos, experimentarían problemas de rendimiento, a nivel de contenedor web. El uso del Servlet de acceso puede crear en algún momento cuellos de botella que provienen, entre otras cosas de las siguientes razones:

- **El modelo de ejecución de Servlet**, especialmente si el cuello de botella del Servlet se apoya sobre un Java Server Pages (JSPs) para la presentación lógica.
- **La latencia creada por la infraestructura del conector**, como JNI. En algunas aplicaciones de la JVM, por ejemplo, el JNI llama forzosamente a la JVM (Máquina Virtual de Java) para crear los pointers para asignar los procesos de plataforma específica pedidos. Algunos pointers ocupan un amplio espacio de memoria y la JVM necesita actualizarlos continuamente para evitar la recolección continua de las áreas de memoria ya no usadas por los hilos (garbage collection), mientras éstos estén aún activos. Esto crea sobrecarga sobre la JVM, lo que significa más CPU y una mayor utilización de los procesos Java en los cuales la JVM se está ejecutando.

Estos problemas no tienen nada que ver con la infraestructura grid. Pueden presentarse aún cuando la aplicación no esté corriendo en un grid. Es necesario estar consciente de que se tendrá que poner a punto el servidor de aplicación una vez que bajo las nuevas condiciones se haya desplegado el grid.

Hay otros problemas que provienen de la interacción del servidor de aplicación y la infraestructura grid. Como por ejemplo:

- **Desbordamiento de seguridad**, entre el servidor de aplicación y la infraestructura grid. Los grids son un monstruo de seguridad y las llamadas JNI no siempre son piden para ser autenticadas. Como solución, a veces, aplicación tiene que loguearse en la infraestructura grid cada vez efectúa una llamada al connector. Esto puede reducir considerablemente la velocidad de las cosas en presencia de sobrecarga.
- **Latencia de la red**, entre el servidor de aplicación y la infraestructura grid, especialmente en despliegues geográficamente dispersos.

Cuando algunos de estos problemas, aparecen al mismo tiempo, hacen que el ejercicio de habilitación de grid en general también sea también complicado y demasiado costoso como para que valga la pena el esfuerzo. Muchas veces es bueno considerar la posibilidad de desplegar los módulos de plataforma específica de la aplicación como una aplicación regular *monolítica* o *modular* (aplicar el que sea), o bien volver a describirlos como un juego de componentes compilados J2EE y tener como perspectiva un grid basado en SOA.

Aplicaciones Database-centric Las aplicaciones *Database-centric* (centradas en bases de datos) se convirtieron hoy en día en el estándar de facto del paradigma cliente-servidor. Herramientas tales como PowerBuilder, Oracle2000, Pacbase, Progress, y otras fueron utilizadas para crear aplicaciones *monolíticas*, con grandes colas de impresión y aplicaciones homogéneas que requieren de un lenguaje propietario y herramientas especializadas para su comprensión.

Algunos de estos productos fueron manejados para adaptar el nuevo paradigma distribuido y brindaron como resultado los híbridos de habilitación web que ahora se conoce como aplicaciones *Database-centric*. Mientras algunos

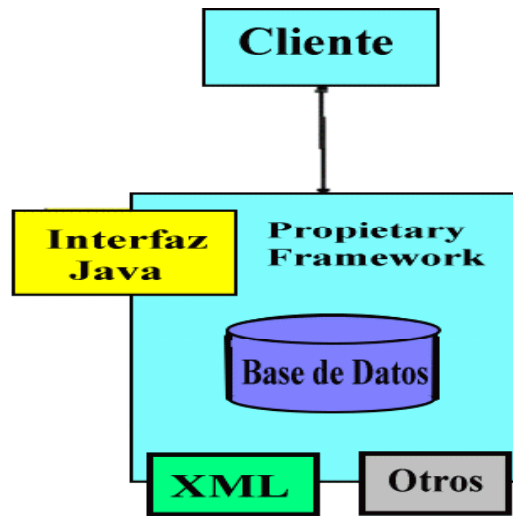


Figura 3.15: Modelo arquitectónico más común Database-centric.

vendedores exigían la reingeniería y adecuación para la Web de sus productos para que sean verdaderamente distribuidos, otros, bajo los nuevos esquemas, permiten que los viejos esquemas cliente-servidor y la arquitectura monolítica permanezcan intactos.

Esto da a entender que incluso la mayoría de los vendedores inventaron sus propios lenguajes para demostrar que las estructuras complejas apuntaban a facilitar lo que en esa época se denominaba “Rapid Application Prototyping and Development”.

Sin tener en cuenta el valor técnico de estos productos. Los vendedores necesitaban conservar este capital intelectual por una simple razón: se ha invertido mucho dinero en su desarrollo y necesitan aprovecharlos. Esto significa que las aplicaciones *Database-centric* serán desarrolladas en torno a ellos por mucho más tiempo.

Las implementaciones más comunes de aplicaciones *Database-centric* se desarrollan en torno a una infraestructura (entorno) propietario. En muchos casos ésta infraestructura ha sido “arreglada” para soportar Java, XML, JMS sobre algún otro nuevo estándar popular de la industria. La figura 3.15 de la página 104 ilustra el cometido más común de esta arquitectura.

En muchos casos la aplicación y la base de datos están juntas en un solo

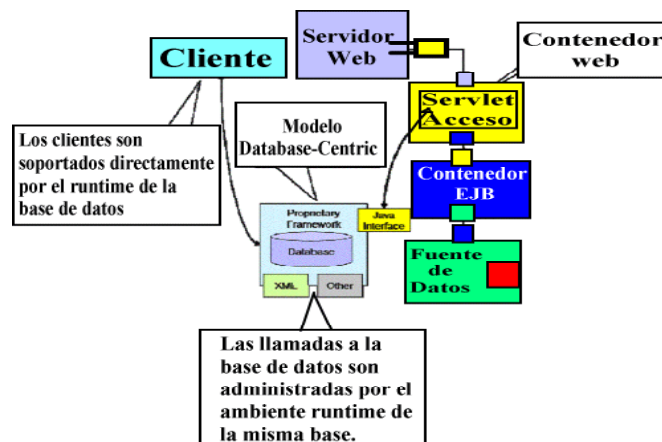


Figura 3.16: Escenario típico de habilitación web Database-centric.

paquete y no hay ninguna diferencia entre en la lógica comercial y la lógica operacional de la base de datos. Debido a esto, los módulos de apoyo adicionales no interactúan recíprocamente en principio con la aplicación original.

Se construyen escenarios para que las aplicaciones *Database-centric* sean usualmente hechas a través de un contenedor Web como lo muestra la figura 3.16 de la pag 105.

La estrategia es muy similar a la utilizada para aplicaciones *Servlet-centric* (centradas en servlet). Involucra la escritura de un Servlet de acceso que accede al entorno propietario a través de módulos adicionales de soporte Java como clases dependientes o a través de archivos XML.

Tratamiento Como Una Aplicación Monolítica La manera más fácil de habilitar en un grid una aplicación *Database-centric* es tratarla como una aplicación *monolítica*. Para ello se necesitará implementar el modelo de despliegue mostrado en la figura 3.8 de la página 93.

Pero hay características más interesantes acerca de las aplicaciones *Database-centric* que pueden proporcionar maneras más eficaces de desplegar ciertas aplicaciones en un grid.

Virtualización de Datos Las aplicaciones *Database-centric* no hacen uso de las bases de datos solamente para guardar datos sino que también la utilizan para mantener información referente a la configuración, datos de workflow e incluso la representación de los metadatos. Esta dependencia de las bases de datos es muchas veces tan fuerte que resulta casi imposible separar la base de datos del entorno de tiempo de ejecución (runtime). En algunos casos, la aplicación se ejecuta indudablemente sobre el motor de la base de datos.

En casos donde el runtime de la base de datos es el runtime de la aplicación, lo que puede ser virtualizado no es la lógica de negocios, sino los datos. Esto significa pasar de un grid de datos a un grid computacional como en los escenarios previos.

Al virtualizar una aplicación *Database-centric* en un grid de datos, se estaría virtualizando indirectamente el runtime de la aplicación y así la aplicación misma. Todos los datos de la aplicación necesitan para ejecutarse estar disponibles localmente en todos los nodos del grid y cada vez que la aplicación se modifique, los cambios se propagarán automáticamente.

Resumidamente lo que sucede es que se consigue la independencia de localización para las peticiones que van a la base de datos mientras los datos se propagan por todo el grid. Para el programa peticionante los datos parecen estar localmente disponibles todo el tiempo, cuando en realidad pueden estar localizados en cualquier parte del grid. Así, lo que se ejecuta como un trabajo batch de instancia simple es el broker -la interfaz java y una unidad de cliente- mientras los datos son virtualizados por la infraestructura grid.

Dependiendo de cómo desde ahora en adelante la lógica de negocios es definida, puede que sea posible la virtualización de los datos y la lógica de negocios como se muestra en la figura 3.17 de la página 107 .

Notar que la virtualización de datos y la lógica comercial en un grid de datos puede lograrse únicamente si la lógica de negocios corre en procesos activados por el motor de base de datos, que es el caso de las mayoría de los entornos propietarios.

Si, por el contrario, la lógica comercial puede ser activada desde un proceso externo como por ejemplo una unidad de entrega de trabajo, entonces es posible separar realmente los datos de la lógica comercial en lo que sería una combinación de un grid computacional para la lógica comercial y un grid de datos para la base de datos. Este método, sin embargo, puede no ser una solución factible en algunos casos, porque acarrearía demasiada complejidad

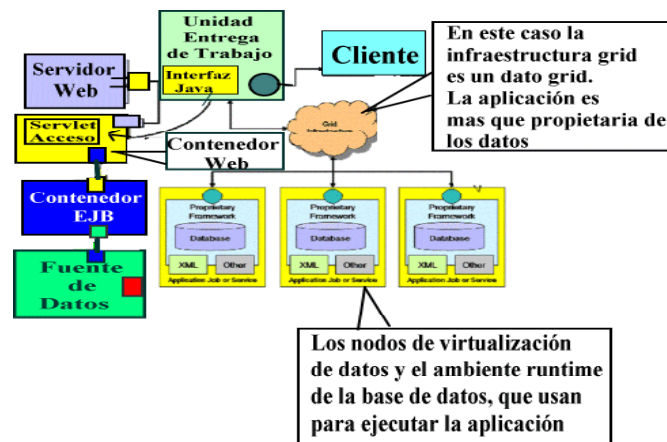


Figura 3.17: Escenario de virtualización de datos para aplicaciones Database-centric.

en el modelo de despliegue.

Riesgos Las aplicaciones *Database-centric* normalmente tienen la característica de ser muy pesadas en términos de uso de CPU y memoria. Esto se ve especialmente cuando el runtime de la base de datos también ejecuta aplicaciones.

En algunos casos, el ejercicio de despliegue del grid no puede trabajar como se esperaba debido a la sobrecarga creada por los mismos datos del grid, más la sobrecarga creado por un recurso ocioso del ambiente runtime de base de datos. En otros casos, siempre que sea posible separar los datos y la aplicación, la situación se reduce al caso de las aplicaciones *monolíticas* y al que se le será aplicado los riesgos de estas.

3.5 4^o, 5^o y 6^o Grado de Adopción

3.5.1 Servicios (4^o Grado)

Objetivos

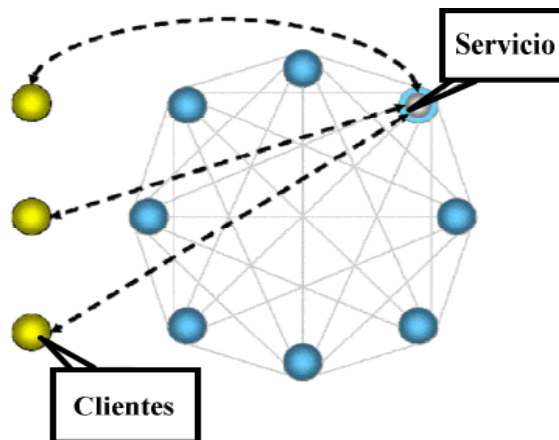
El objetivo planteado es lograr que la aplicación se ejecute como un servicio o como una colección de servicios que se convertirán en blanco para algún formulario de llamadas a procedimientos remotos.

Lo que califica a este modelo de ejecución como un grado de habilitación de grid es que el servicio (o servicios) debe haber alcanzado las características de los anteriores grados de habilitación de grid, tanto como hacer posible su ejecución en múltiples nodos seleccionados por la infraestructura grid. Lo que se desea que haga un servicio es que, pueda ser invocado muchas veces sin seguir un esquema de planificación por lote, programa de iniciación y operaciones de gestión interna de terminación después de cada llamada.

Por ejemplo, una aplicación que tiene que completar una secuencia de tareas como la verificación de licencias e inicialización de las utilidades de las estructuras de datos (talvez varias GB de datos en memoria) desde fuentes de datos externos, cada vez que se invoquen beneficiaría enormemente ejecutarlas como un servicio ya que estas tareas serían completadas de una sola vez. Incluso siendo un servicio, una instancia del mismo puede ocuparse de varias peticiones concurrentes independientes.

Notar que se está hablando de un servicio en el sentido más genérico de la palabra. No se está haciendo referencia específicamente a " *Grid services: servicios Grid*" o a " *Web services: servicios Web*" porque estos términos tienen un significado específico como estándares abiertos y otras implementaciones de arquitecturas orientadas al servicio que predatan OGSA y *Servicios Web*.

El grado de habilitación se enfoca en la transición de los lotes a una arquitectura orientada al servicio. Este grado de adopción es una continuación del segundo grado, es decir, del **Independent Concurrent Batch**, y no del tercero que se ejecuta como un trabajo batch divisible. Se asume que el cliente subdivide el trabajo y lo despliega sobre múltiples instancias del servicio. Esto es mostrado en la figura 3.17 de la página 109.

Figura 3.18: 4^o Grado de adopción.

Características y Beneficios

En adición a las características del segundo grado de habilitación, este grado tiene las siguientes propiedades:

- El cliente utiliza la infraestructura grid para invocar al servicio. La infraestructura grid ejecuta el servicio como una clase, subrutina, librería, o un DLL.
- El cliente y el servidor son acoplados de forma no intensa. Esto permite que el servicio pueda ser compartido entre los clientes independientes.
- El servicio puede mantener el estado entre las llamadas.
- El tiempo extendido de inicio de un servicio no es un problema, con tal de que la ejecución de los mismos sea extendida, usado mucho tiempo por cada corrida, y preferentemente iniciado antes del primer uso.

Los beneficios más importantes son:

- **Rapidez de resultados:** Las peticiones pueden ser rápidamente manipuladas cuando cualquier operación interna de inicialización de programa ya están siendo realizadas o antes de la primera petición.

- **Reducción en la consumición de recursos o incremento de la capacidad:** Los recursos intensivos de programas de operaciones internas de inicialización pueden ser usados una sola vez y explotados por mucho tiempo.
- **Statefulness o statelessness:** El servicio puede mantener información basada en los usos previos por el cliente/s o puede no mantener información del estado en su totalidad.
- **Capacidades multiusuarios:** Cada instancia del servicio puede ser habilitada para administrar múltiples clientes.

Escenario Básico

Para este tipo de habilitación, el escenario básico es un programa que toma parámetros en línea de comandos y usa archivos y/o bases de datos como los especificados en los parámetros. Con esto ya se reúnen los atributos requeridos por el primero y segundo grado de habilitación, y muchos de aquellos necesarios en el tercer grado.

Usando la terminología de cliente y servidor (ver figura 3.8 de la página 93) este programa se convertirá en una colección de subprogramas del servidor que pueden ser ocupados por cualquier otro cliente. Un cliente puede crear una nueva formula comercial haciendo provecho de muchos servicios al mismo tiempo.

En este caso, la noción de aplicación conocida hasta ahora por nosotros tomará el nombre de aplicaciones morph dentro de servicios.

Conversión a Servicio

Un servicio es un programa cuya ejecución no depende de una llamada a nivel de cliente. De esta manera, por cumplir tan simple requisito, virtualmente cualquier programa puede convertirse en servicio. No obstante a ello, hay toda una logística en el proceso de conversión a servicio.

Para convertirse en servicio, una o más capacidades del programa deben convertirse en funciones o subrutinas llamables externamente. Las entradas y las salidas deben ser especificadas como parámetros y el programa debe además mostrar una interfaz al mundo externo.

Esto puede lograrse de varias formas. Sería interesante ver como es alcanzado mediante la utilización de la especificación *Web Services Resource Framework* (WSRF) la cual cumple con todos los requisitos listados anteriormente y proporciona un buen mecanismo para definir el statefulness en los servicios.

Además de los requisitos externos ya antes mencionados, el programa mismo es el que deberá cumplir con la serie de características que también ya han sido citadas

Reusabilidad

Una vez cargadas, las funciones públicas deberían, y prácticamente, ser reusables. Si no lo son, la infraestructura grid necesitaría cargar y descargar el programa entre los diferentes usos. Esto forzaría a que la aplicación ejecute operaciones internas de inicialización y terminación para cada uno de los usos, y es lo que justamente se intenta evitar.

Como mínimo, los servicios deberían ser “reusables continuamente”. Esto significa que la infraestructura grid debería pasar las peticiones del cliente, una tras otra a las funciones del servicio.

Se debe lograr la verdadera reusabilidad de las funciones o por lo menos asegurar los hilos si correrán bajo un middleware de grid que requiere de reusabilidad de serie. El *Globus Toolkit v3.0*, por ejemplo requiere reusabilidad continua.

Servicios Individuales y En Serie

Al crear un servicio, es una buena idea preparar tanto, un servicio individual para manejar un único ítem, y un servicio en serie, que pueda ejecutar el servicio para varios ítems en una única petición. Si se lo puede hacer, permitirá al cliente evitar el desbordamiento de ítems terminándolos en una única demanda.

Servicios Reusables vs. Ejecuciones Discretas de Programas

Los servicios reusables requieren que las variables de ambiente sean fijadas antes de la inicialización del servicio y no se modifiquen entre las distintas invocaciones.

Si la aplicación original obtuvo lo que son, en efecto, los parámetros por demanda desde el ambiente, estos deberían ser convertidos en parámetros a nivel de función. Las variables de ambiente que no cambian entre las distintas demandas deben permanecer como tal.

Los procesos, *stdin*, *stdout* y *stderr* no son modos típicos por los que un servicio recibe una entrada desde un cliente o envía una salida o retorna información de error. El intercambio de información del programa usuario que ha utilizado esos mecanismos debería y usualmente debe remplazarlos por otras alternativas, tales como cadenas de parámetros. Usualmente se aconseja algún software de infraestructura grid, como el *DataSynapse Grid Server* que proporciona *stdin* para la transferencia desde el cliente y *stdout* y *stderr* para las respuestas desde el servicio.

A menudo, una cierta cantidad de información de error necesita ser regresada al cliente, mientras más información se retenga, mayor será el análisis por el servicio o administradores grid.

Instancias Múltiples

Este grado de adopción de grid permite que múltiples instancias independientes de rutinas de servicios sean cargadas concurrentemente en la misma máquina o en servidores diferentes. Es muy similar al segundo grado de adopción de grid presentado anteriormente.

Generalmente, se debe considerar la posibilidad de que en la infraestructura grid se puedan proyectar varias instancias de un mismo servicio al mismo tiempo. Además, se debería cumplir con todos los requisitos impuestos por el segundo grado de adopción al estructurar una aplicación cuya ejecución será como un servicio. Sin embargo, hay situaciones en las que esto no es deseable.

Tener presente que si un servicio es configurado para retener cualquier tipo de información de estado entre las llamadas a servicios este debe, ejecutarse solo en ambientes como OGSA, donde pueda ser certificado, que cada petición

de servicio desde el cliente es enviada a la instancia del mismo servicio, o bien se debería almacenar esta información en alguna localización accesible por las otras instancias. Cada instancia debe saber como recibir la información de estado desde esa localización, si la demanda del servicio anterior fue utilizada por otra instancia.

Usuarios Múltiples

La existencia de múltiples clientes para una instancia de un servicio puede no ser soportada, o bien, ser obligatoria, dependiendo del ambiente runtime de la infraestructura grid. Si no es soportado, sólo un usuario utilizaría cada instancia del servicio sin considerarse la carga de trabajo. Si por el contrario es obligatorio, entonces el diseñador de servicios debe lograr que cada instancia del programa sea capaz de manejar múltiples usuarios.

Cuando un programa administra múltiples usuarios, puede que el tratamiento de seguridad tenga que cambiar. El servicio puede necesitar enviar o guardar cualquier tipo de información acerca del usuario previo aunque éste esté autorizado a lanzar otra petición.

La seguridad en el ambiente debería controlar si un determinado cliente puede o no llamar a un servicio, pero no determinar si el programa cliente está autorizado a lanzar peticiones de servicios en nombre de un usuario X sin considerar el número de cuenta del usuario Y. El propio servicio es quien debe ocuparse de esa autorización.

Dependiendo de los recursos usados por el servicio puede que sea necesario para el servicio cambiar el contexto de seguridad, es decir, un ID de usuario cliente para los distintos usuarios, en lugar de usar un ID genérico de usuario (como el root, por ejemplo).

Otro aspecto importante a considerar en el caso de que una instancia soporte múltiples usuarios, es el tema de la concurrencia. Si la infraestructura grid o ambiente de servicio puede garantizar que sólo una petición del servicio sería procesada en cualquier momento dado por cada instancia del servicio, entonces la reusabilidad en serie sería suficiente. Por otra parte, el servicio debe estar lo suficientemente seguro. Estar seguro significa que el programa debe resguardar cada uno de los hilos separadamente sin tener en cuenta quién los ha iniciado. Esto quiere decir que un programa debe brindar la posibilidad de que uno de sus propios hilos sean entregados por el ambiente runtime.

Esto es lo que pasa en productos tales como WAS y CICS . En el caso de WAS, cuando un nodo agente percibe que un bloque del hilo está finalizando, inicia un nuevo contenedor Web o hilo ORB y lo entrega tan rápido como pueda a la JVM.

Licenciamiento

Si el programa requiere una licencia, entonces se debe considerar la posibilidad de usar nuevos modelos de licencia, si es que se pretende ejecutar a éste como un servicio.

El modelo de licencia usado comúnmente para servicios es “*per-use*” . Se debe determinar que significado adquiere el “use”. Esta puede ser una petición de invocación, o un conjunto de peticiones de servicios relacionados con un cliente.

Definición de Clientes e Interfaces

Se debe proporcionar un escenario ejemplo, o una arquitectura de concepto de prueba, de un programa cliente que llama a los servicios del servidor de aplicaciones. El escenario no debería mostrar sólo uno de los usos de servicios. Si no que, mostrar apropiadamente las sucesiones de uso y las series de servicios usados.

Entonces se debería construir un escenario y cuando esté listo para probar, el equipo de integración y despliegue debería adaptar el modelo final de despliegue para ajustar el seleccionado runtime del ambiente.

Se tendrá que hacer lo mismo con cada uno de los clientes para que ellos también puedan adaptar su modelo de despliegue a sus ambientes grid. Esta práctica debería convertirse en un protocolo que se inicia con la unión de los servicios y el despliegue, y que permanece sin alteraciones para todos los clientes.

Acoplamiento Leve Entre los Clientes y los Programas del Servidor

El cliente debería utilizar la infraestructura de grid para encontrar, lanzar y conectar al servidor los servicios de aplicación de una manera segura. Tam-

bién podría utilizarla para enviar las peticiones de servicios individuales, no obstante, la infraestructura grid presenta un leve o común acoplamiento entre los clientes y los programas del servidor.

Típicamente la proporción de las peticiones de un servicio de un cliente a través de la infraestructura grid tardarían menos de un segundo.

Una llamada de servicios a través de la infraestructura grid es relativamente cara, análoga a la de los *Servicios Web*, es comparado con el envío de un mensaje a través de un **socket** o interfaz de programación **named-pipe** en algún formato de propiedad. El alto volumen y las interacciones fuertes entre el cliente y el servidor deberían evitarse. Si desgraciadamente son inevitables la aplicación debe manejarlos a través de un mecanismo relativamente eficiente (comparado con el de los *Servicios Web*), como CORBA o llamadas a procedimientos remotos Java sobre IIOP .

Esto es apropiado utilizar la infraestructura grid para encontrar, lanzar y conectar el servicio de aplicación de una manera segura y después utilizar un mecanismo adicional para manejar las altas tasas producidas por las intercomunicaciones entre el cliente y el servidor.

Una manera de reducir la cantidad de peticiones de servicios que atraviesan la red es empaquetar varias peticiones y después invocar un servicio en serie (ver en “conversión a servicios”) que manipule varias peticiones que llegan simultáneamente.

3.5.2 Servicios Paralelos (5^o Grado)

Objetivo

El objetivo en este grado de habilitación es que una aplicación se ejecute como una colección de servicios que le permita a los usuarios subdividir el trabajo entre las distintas instancias.

Características y Beneficios

Este grado de habilitación de grid combina las características de ejecución del Batch Paralelo y las de los servicios (3^o y 4^o grado de habilitación), éste es ilustrado en la figura 3.19 de la página 116.

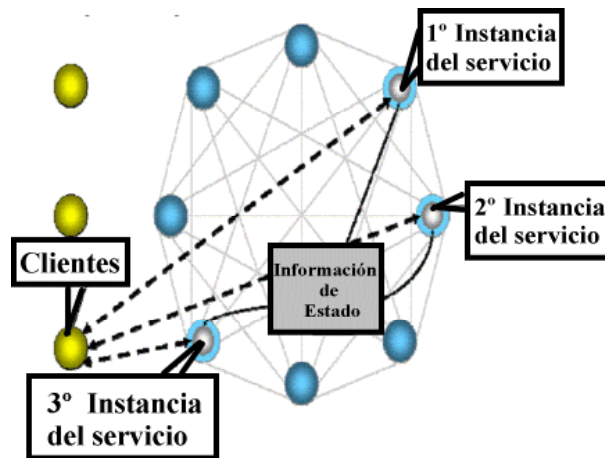


Figura 3.19: 5º Grado de adopción de Grid.

Esta fusión del 3º y 4º grado de adopción de grid provee beneficios considerables. Por ejemplo, el tiempo de respuesta mejora las deficiencias de ambos grados, el paralelismo y el hecho de que los servicios sean iniciados una sola vez, precede preferentemente al procesamiento de la primera demanda (aunque este no siempre es el caso).

Otro de los beneficios destacables de la combinación del paralelismo y de la ejecución tipos de servicios, es que ahora los clientes pueden explotar los servicios de nuevas maneras.

Paralelismo Basado En el Estado

La técnica del “paralelismo leve” es usada para ejecutar los trabajos batch paralelos basados en la transmisión de diferentes parámetros o bien, usada para la ejecución de instancias del mismo subtrabajo. Esto a veces es denominado como paralelismo paramétrico.

Para el caso de los servicios paralelos, hay un tipo diferente de paralelismo denominado paralelismo basado en el estado. Aquí la ejecución paralela está regida por información de estado en lugar de la transmisión de parámetros. En los ambientes paralelos basados en los estados cada instancia del servicio mantiene su información de estado, por lo general en cada instante tiene uno diferente, más adelante se podrá entender porque ésto es importante.

Multidifusión de Clientes

Gracias al hecho de que, para los clientes todas las instancias del servicio son diferentes (todos con estados diferentes), pueden multidifundir sus parámetros. Al usar la multidifusión, un cliente invoca todas las instancias del servidor con el mismo conjunto de parámetros, entonces el servidor procesa cada una basándose en el estado y la información que ya tiene. Esta es la manera en que cada instancia puede cambiar de estado y devolver un resultado apropiado.

Consideremos el ejemplo de la teoría de los números. Se tiene 100 instancias de servicios del servidor, donde cada una verificaría si los números entrantes son divisibles por listas de números primos de las mismas instancias.

La primera instancia del servicio tiene la lista de los primos entre el 1 y 1000. La segunda instancia del servicio tiene la lista de todos los primos entre el 1001 y el 2000 y así sucesivamente. Al enviar un número grande a las 100 instancias de una sola vez (en multidifusión), este puede ser chequeado para ver si el parámetro es divisible por cualquier primo menor que 100.000.000 en “aproximadamente” 1/100th, que sería el tiempo que tomaría una instancia en hacer todo el trabajo.

Hay una razón por la que se dice que es “aproximadamente”, toma menos tiempo que en batch paralelo, porque cada instancia del servicio no necesita cargar o calcular su lista para manejar cada una de las peticiones. Por otro lado, la infraestructura grid consume un bit más de tiempo despachando las peticiones de servicios a las 100 instancias del servicio que justo habrían en ese momento .

En el ejemplo anterior el mismo parámetro fue multidifundido a todos los servidores de una sola vez. Existen además otros tres modelos de invocación conocido como Eachcast, Anycast, y Unicast.

Eachcast

Eachcast significa que un cliente puede enviar en particular, un conjunto de parámetros a cada instancia el servicio a través de la infraestructura grid.

En el ejemplo anterior esta es la manera en que cada una de las 100 instancias del servicio es informada de su responsabilidad.

Anycast

Anycast significa que el cliente puede tener un trato con la infraestructura grid que fuera de un número amplio de peticiones de servicios entre un número pequeño de instancias de servicios. Esto es usado especialmente cuando todas las máquinas de servicio no son iguales de rápidas.

Digamos que el cliente en el ejemplo anterior utilizó los servicios para chequear primeramente el caso relativo a 1,000,000,000. Se podría haber dividido el rango extendido en 100 partes iguales y aplicar "Eachcast" nuevamente, dándole a cada instancia una responsabilidad adicional igual. Sin embargo, es inseguro asumir que las máquinas son iguales de rápidas y que están dedicadas al servicio.

Unicast

En lo desarrollado hasta ahora, el cliente no solo tiene acceso a muchas instancias del servicio. Unicast permite al cliente enviar una petición a una instancia particular del servicio, que puede acelerar el proceso al permitir al cliente tomar las decisiones de una mejor manera.

En el ejemplo, si el número a ser chequeado es menor a 1.000.000.000, el cliente puede enviarlo a una instancia particular del servicio responsable del rango en el que ese número está.

Otras Consideraciones

Anteriormente se ha dicho que este grado de habilitación es una fusión del segundo y tercer grado de adopción que ya fueron descriptos en las secciones previas. Se puede seguramente también asumir que los requerimientos técnicos de esos dos grados de habilitación son aplicados en consecuencia a este grado

El caso de estudio ofrece una visión interesante de las oportunidades disponibles a explotar al adaptar un grid en este grado. Sin embargo se necesita ser consiente de que, en muchos casos, tendrá que modificar su producto para poder ejecutarlos como servicios paralelos en ambientes grid.

En adición a los beneficios descriptos en las secciones previas, necesitará además tener en cuenta ciertos problemas al decidirse por la implementación

del 5^o grado de adopción de grid en sus productos.

Interferencia Durante la Inicialización y Finalización

Al tratarse de servicio paralelos, la posibilidad de inicialización de varias instancias de un servicio, en nombre de un mismo usuario, puede ocurrir concurrentemente pero no necesariamente en sincronización.

Se tendrá que tener esto en cuenta si las aplicaciones necesitan acceder a los recursos en los sucesivos procedimientos.

En adición a la interferencia potencial que puede ocurrir debido a la inicialización concurrente, necesitará seguir detenidamente los problemas relacionados con locking, hotspots y deadlock. También, tener en cuenta que mecanismos de caching en los recursos back-end pueden agravar la interferencia y finalizar siendo más dañado de lo que se espera.

La interferencia en la inicialización o finalización es muy probable en arquitecturas totalmente integradas. Cuanto más compleja sea la arquitectura, más complejo será el esfuerzo para habilitarla. En general, un entorno débilmente acoplado en las arquitecturas integradas es una buena elección. Pero este caso de acoplamiento débil puede convertirse en un problema, dado que probablemente no todos los elementos de una arquitectura integrada pueden convertirse a servicios paralelos.

Hay dos estrategias para evitar la interferencia en la inicialización o finalización.

Selfish Instances La primer estrategia es para tener casos de *Selfish instances* (*Instancias Únicas*). Aquí cada instancia envía una petición importante de recursos para conseguir toda la información de inicialización de una sola vez. Esta utiliza todos los recursos disponibles mientras lo hace y no permite incluso que ninguna otra instancia pueda ser iniciada.

Polite Instances La segunda estrategia permite a una instancia compartir los recursos apropiadamente, al enviar peticiones de recursos discretas (pequeñas) para reunir la información de inicialización.

Adoptar una o otra estrategia dependerá de que medios de inicialización

se usen para cada caso. Esto también repercute en el comportamiento actual de un ambiente de instancia múltiple. Los problemas que se presentan en una instancia que completa su inicialización rápidamente al ejecutarla exclusivamente, pueden no ser los mismos que al ejecutar 100 instancias en paralelo. En los casos, en los que se considere este grado de adopción, siempre piense en el guión del escenario del peor caso donde centenares de instancias corren al mismo tiempo.

Definición de Clientes e Interfacesaaaa

Los despliegues típicos de servicios paralelos siguen el modelo de despliegue mostrado en la figura 3.8 de la página 93. Para este grado de adopción, el programa cliente o unidad de entrega de trabajo, tendrán que ser desarrollados en la mayoría de los casos desde el principio. Mientras se hace esto, se tendrá que tener en cuenta algunas variaciones en los servicios, clientes y despliegues de middleware.

Por ejemplo, la infraestructura grid puede proporcionar solo un subconjunto de paradigmas de invocación de servicios Unicast, Multicast, Eachcast, y Anycast. En un extremo el servicio puede ejecutarse con una sola instancia o con middleware que solo soporta Unicast. Al otro extremo, puede haber miles de instancias de clientes, cada cliente usando potencialmente una, algunas o todas de las miles instancias del servicio.

3.5.3 Programas Paralelos Completamente Acoplados (6° Grado)

Objetivo

El objetivo en este grado de adopción es poder ejecutar la aplicación como una colección de servicios que se comunican con cada uno de los otros, y también con el consumidor del servicio. Puede haber muchas instancias de cada servicio ejecutándose en el mismo momento.

Este grado de adopción constituye la generalización de *Grid Computing*. Anteriormente las aplicaciones grid fueron consideradas como tal solo cuando se ejecutaron como programas o servicios paralelos completamente acoplados. Tradicionalmente éste ha sido el dominio de las aplicaciones especializadas en

ingeniería, física y modelado matemático y biológico.

Características y Beneficios

La principal característica de los programas paralelos completamente acoplados es la comunicación intensa entre los clientes y los servicios y como así también entre los servicios mismos.

Técnicamente este grado de adopción de grid debería ser dividido en dos prácticas: uno para la comunicación intensiva de los trabajos batch paralelos fuertemente acoplados y otro para los servicios.

Los problemas aplicados por los trabajos batch y los servicios son muy similares excepto para aquellos problemas que se relacionan con la conversión de un programa batch en un servicio (ver en Servicios y Servicios Paralelos). Esto nos permite ver que no se trabaja con un modelo estándar formal de física nuclear donde cada tipo de partícula es equilibrada con otra.

El Modelo de Ejecución Simple MPP

Hay muchos modelos de ejecución para servicios paralelos fuertemente acoplados. Históricamente el modelo de ejecución ha sido determinado en gran parte para extender al software y al hardware ha sido el dominio exclusivo de la máquinas SMP con propósitos complejos y caros.

Lo que se busca es un modelo de ejecución que permita a las aplicaciones ejecutarse como servicios paralelos fuertemente acoplados en hardware heterogéneo y barato. Es decir lo que se pretende es adecuar el grid a las personas.

El modelo de ejecución es también conocido como MPP basado en nodos de simple procesador. Este cuenta con mensajería y una infraestructura de red para extender el trabajo a través del espacio de dirección adicional formado por el grid. Este modelo de ejecución es mostrado en la figura 3.20 de la página 122.

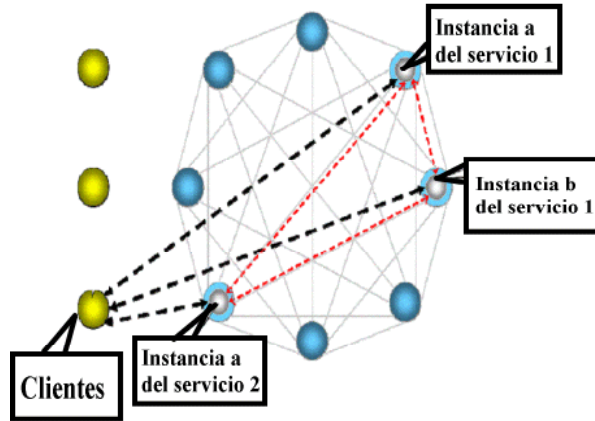


Figura 3.20: 6º Grado de adopción de Grid

Paralelismo Nativo

El tipo de paralelismo requerido para que una aplicación se ejecute como una colección de servicios paralelos fuertemente acoplados es denominado paralelismo nativo. Es diferente al paralelismo redundante y el paralelismo basado en el estado descrito para el 2º y 4º grado de adopción.

En los casos del 2º y 4º grado la idea era ejecutar código de serie en paralelo en varias máquinas. En el caso del paralelismo nativo es la aplicación quien es nativamente paralela o paralela de nacimiento. Esto significa que el paralelismo es incluido en el código fuente. Hay muchas infraestructuras y librerías disponibles para escribir código paralelo. La mayoría de ellos son escritos en C o Fortran.

Este grado de adopción es muy diferente a los cinco anteriores ya vistos hasta ahora. Se tendrá que estar consiente que va a ser muy caro y va consumir mucho tiempo efectuar una extensión para que la aplicación no nativamente paralela alcance este grado de adopción. Básicamente tendrá que tomar y volver a escribir entre el 70 % y 80 % de código, en cualquier caso, esta práctica sería buena solo para empezar desde el principio.

El paralelismo redundante y el paralelismo basado en el estado son principalmente modelos de despliegue. Por otro lado, el paralelismo nativo es un modelo de programación. Esto significa que el paralelismo nativo es un

lenguaje dependiente.

Paralelismo Nativo Usando MPI

MPI soporta la interfaz de comunicación de mensajes. MPI es un estándar abierto desarrollado por el Forum de MPI, que puede visitar en <http://www.mpi-forum.org>. La versión actual del estándar es el 2.0.

El estándar proporciona comunicación punto a punto, operaciones colectivas, grupos de procesos, contextos de comunicación y tipologías de procesos. También incluye los enlaces para Fortran 77 y C. y administración medio ambiente, consulta y perfilado de interfaces.

Paralelismo Nativo en Java

Digamos que el problema del paralelismo nativo en java es mucho más difícil que en cualquier otro lenguaje. No alcanza con ir y escribir solamente una infraestructura o conjunto de librerías para soportar el paralelismo de los bucles **do-while** porque no se tienen acceso a la CPU desde donde se está. Lo que se tiene es una JVM entre la CPU y el código.

De esta manera, el paralelismo nativo en Java debe ser implementado de una forma diferente.

Lo que se necesita tener presente respecto del paralelismo nativo en Java es que la mayoría de las instrucciones para activar el procesamiento paralelo serán comandos reales a la JVM inversamente a instrucciones a nivel de CPU como C y Fortran.

Por ejemplo, sobre ambientes simples de JVM, el solo hecho de paralelismo nativo puede tomar ventaja si la habilidad de la JVM son los *multi-thread*. Aún si la máquina es una caja *multi-way*, si hay una única JVM, lo mucho que se puede hacer es asignar bucles a diferentes hilos y esperar que el sistema operativo distribuya los procesos de JVM uniformemente entre las CPUs disponibles.

Cuando el ambiente es un ambiente de multi-JVM se puede distribuir los componentes de la aplicación para ejecutarlas sobre diferentes JVMs como aplicaciones discretas. Pero para hacer esto los componentes de la aplicación tienen que estar débilmente acoplados y la arquitectura debe proveer una

sincronización mínima entre componentes.

Adicionalmente, los ambientes *multi-JVM* requieren las APIs especializadas para la comunicación inter-JVM. El JMS y WebSphere MQ no son exclusivos para los propósitos de este grado de habilitación de grid pero las APIs especializadas pueden estar basadas en cualquiera de las dos.

3.6 En Resumen

El sexto grado de adopción o habilitación de grid, constituye un mapa de estrategias u objetivos para adoptar el paradigma de *Grid Computing* en la ejecución y programación.

Para los tres grados de adopción más bajos la aplicación ideal debería requerir modificaciones en sus archivos de despliegue solamente. Por otro lado, los tres grados de habilitación más altos presentan tres niveles diferentes de adopción, una nueva ejecución y programación. El concepto de ejecución como servicio requiere de varias capacidades y no solo la modificación de los descriptores de despliegue sino que también del código fuente de la aplicación.

Esta separación entre despliegue y código fuente es definida por el costo del esfuerzo para lograr un grado de adopción de grid. En algunos casos, el costo de llevar adelante un proyecto para habilitación de grid en una aplicación existente puede ser también muy elevado comparado con los beneficios comerciales que se pueden conseguir en el mercado. Muchas veces el costo de escribir una aplicación nativamente paralela puede ser menor que la habilitación de código existente que se ejecuta a un grado más bajo de adopción.

Dado el impacto que los problemas comerciales tienen sobre las habilitación de grid, resulta crucial tratar de minimizar el esfuerzo al tratar el problema en términos de despliegue vs. modificación de código.

El aspecto más importante de los tres grados superiores de adopción es que ellos son implementados de una mejor manera si están basados en los estándares abiertos.

Afortunadamente, los estándares existen, la *Open Grid Services Architecture* (OGSA) y el *Web Services Resource Framework* (WSRF) constituyen la fundación para los estándares de grid e implementaciones grid orientadas al servicio de los tres grados superiores de habilitación de grid.

Bibliografía

- [1] L. Joyanes Aguilar. *Cibersociedad*. Mac Graw-Hill, 1997.
- [2] Viktors Berstis. *Fundamentals of Grid Computing*. IBM, USA, 2002.
- [3] James A. O'Brien. *Sistemas de Información Gerencial*. Editorial Nomos S.A., Argentina, 2003.
- [4] Arun K. Iyengar Marcos Novaes Li Zhang Catherine H. Crawford, Daniel M. Dias. *Commercial Applications of Grid Computing*. IBM, USA, 2003.
- [5] H. Nimrod D. Abramson, R. Giddy Sasic. *The Grid: Blueprint for a New Computing Infrastructure*. IEEE, USA, 1995.
- [6] A. González del Alba Baraja; V. Yague Galaup; L. Joyanes Aguilar. *Impacto de las Tecnologías en la Gestión de los Sistemas de Información en II Congreso Internacional de Sociedad de la Información y del Conocimiento*. McGraw Hill, Madrid-España, 2003.
- [7] J. Nick S. Tuecke I. Foster, C. Kesselman. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Global Grid Forum, USA, 2002.
- [8] R. Gutierrez I. Foster B. Ginsburg O. Larsson I. Lopez, G. Follen. *Using CORBA and Globus to Coordinate Multidisciplinary Aerospace Applications*. NASA, USA, 2000.
- [9] T. Durniak P. Herman J. Karuturi C. Woods C. Gilman J. Barry, M. Aparicio. *Enterprise Distributed Computing Workshop*. IEEE, USA, 1998.
- [10] H. Bivens S. Humphreys R. Rhea J. Beiriger, W. Johnson. *Constructing the ASCI Grid*. IEEE, USA, 2000.

- [11] Q. Peng D.P. Schissel M. Thompson I. Foster M. Greenwald D. McCune K. Keahey, T. Fredian. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. 2002.
- [12] M. Kaufmann. *The Grid: Blueprint for a New Computing Infrastructure*. 1999.
- [13] MIT. *Technology Review*. MIT, USA, 2004.
- [14] N.Ñegroponte. *El Mundo Digital*. Ediciones B, Barcelona-España, 1995.
- [15] A. Keahey K. Kohn S. McInnes S. Parker R. Armstrong, D. Geist Gannon. *The Grid: Blueprint for a New Computing Infrastructure*. IEEE, USA, 1999.
- [16] Colin J. White. *IBM Enterprise Analytics for the Intelligent e-Business*. IBM Press, USA, 2001.

Índice de Materias

- AFS, 36
- Ambiente Distribuido, 3
- ambiente hosting
 - componente de IBM Grid Toolbox, 24
- Batch Anywhere, 73
- bubbled up, 48
- certificado de autoridad, 61
- CICS, 114
- Computación Distribuida, 1
- concepos y componentes
 - trabajos y aplicaciones, 40
- conceptos y componentes, 34
 - almacenamientos, 36
 - Comunicaciones, 37
 - comunicaciones
 - externa, 38
 - interna, 38
 - equipos, capacidades, arquitecturas y políticas, 39
 - software y licencias, 39
 - tipos, 34
 - computación, 35
- configuración de datos, 56
- CORBA, 115
- cpu paralela, 29
 - algoritmos, 29
 - primer barrera, 30
 - segunda barrera, 30
- Data Striping, 37
- database-centric, 103
- DataSynapse Grid Server, 112
- DB2 Information Integrator, 82
- DBMS, 82
- DFS, 36
- DOE
 - DOE Science Grid, 12
- escalabilidad, 35
- estándares
 - Grid Services, 16
 - OGSA, 17
 - OGSI, 18
 - tendencias futuras, 18
 - Web Services, 16
- FTP, 59
- GGF
 - Global Grid Forum
 - Foro Global de Grid, 4
- Globus, 14
- Globus Project
 - Proyecto Globus, 15
- Globus Toolkit, 76
- GPFS, 36
- Grid Computacional, 3
- Grid Computing
 - definición, 67
 - principios, 27
- grid computing
 - componentes, 46

- administración del grid distribuido, 49
- comunicaciones, 51
 - de administración, 47
 - observación, dirección y medición, 51
 - schedulers, 49
 - software de sumisión, 49
 - software servidor, 47
- construcción, 45
- GridSystems, 12
- GSI
 - Grid Security Infraestructure
 - Infraestructura de Seguridad Grid, 12
- GUI, 54
- heurísticas, 42
- IBM Grid Toolbox
 - Toolbox Grid de IBM, 15
- IIOP, 115
- Independent Concurrent Batch, 73
- intragrid a intergrid, 42
 - compatimiento de archivos, 42
- introducción, 1
 - concepto, 1
 - beneficios, 3
- IT, 34
- J2EE, 99
 - Modelo de Programación J2EE, 67
- JDBC, 96
- JVM, 114
- LAN, 78
- Linux, 68
- Máquina Virtual Java, 100
- Mid-grain, 89
- monitoreo del progreso y recuperación, 57
- MPI, 51
- National Institute of Standards and Technology, 2
- NFS, 36
- ODBC, 96
- OGSA, 15, 16, 64, 76
 - Arquitectura de Servicios Grid Estándar, 18
- OGSI, 15
 - Infraestructura de Servicio Grid Estándar, 17
 - Infraestructura de Servicios Grid Estándar, 18
- On Demand, 67
- On Demand Solutions
 - Soluciones Bajo Demanda, 21
- Open Grid Services Infraestructure, 68
- Oracle, 10
- ORB, 114
- Parallel Batch, 73
- Parallel Services, 73
- per-use, 114
- Perl, 94
- perspectiva del administrador, 58
- tilde nador, 64
- perspectivas de usuario, 52
- perspectivas del administrador
 - instalación, 59
 - planeación, 58
- planificación del despliegue, 45
 - organización, 46
 - seguridad, 45
- puggability, 32
- Python, 94

RDBMS, 96
 recursos, 35
 recursos adicionales, 31
 registrarse en el grid, 53
 reservación, 41

 sandbox, 55
 scavenging, 41
 Servicio de Scheduler Distribuido,
 11
 Servicios Web, 68
 SSL, 10

 Table Access Scheduling, 96
 TCP/IP, 10
 teleinmersión, 9
 Tightly Coupled Parallel Programs,
 73

 VPN tunneling, 43

 WAN, 78
 WAS, 101, 114
 Web Services Resource Framework,
 68
 WebSphere Application Server, 100
 WSDL, 16
 WSRF, 76
 Entorno de Desarrollo de Recurso-
 WS, 19

 XML, 16

