Redbooks Paper

Viktors Berstis

Fundamentals of Grid Computing

The purpose of this IBM Redpaper is to provide discussion material about grid computing, concepts, use, and architecture. Grid computing represents unlimited opportunities in terms of business and technical aspects. The audience for this paper are all hungry minds looking for a collection of facts and data about this new and exciting realm.

The following major topics will be introduced to the readers:

- What grid computing can do
- Grid concepts and components
- Grid construction
- The present and the future
- What the grid cannot do

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

What grid computing can do

When you deploy a grid, it will be to meet a set of customer requirements. To better match grid computing capabilities to those requirements, it is useful to keep in mind the reasons for using grid computing. This section describes the most important capabilities of grid computing.

Exploiting underutilized resources

The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid.

© Copyright IBM Corp. 2002. All rights reserved.

There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application.

For example, a batch job that spends a significant amount of time processing a set of input data to produce an output set is perhaps the most ideal and simple use for a grid. If the quantities of input and output are large, more thought and planning might be required to efficiently use the grid for such a job. It would usually not make sense to use a word processor remotely on a grid because there would probably be greater delays and more potential points of failure.

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5% of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

The processing resources are not the only ones that may be underutilized. Often, machines may have enormous unused disk drive capacity. Grid computing, more specifically, a “data grid”, can be used to aggregate this unused storage into a much larger virtual data store, possibly configured to achieve improved performance and reliability over that of any single machine.

If a batch job needs to read a large amount of data, this data could be automatically replicated at various strategic points in the grid. Thus, if the job must be executed on a remote machine in the grid, the data is already there and does not need to be moved to that remote point. This offers clear performance benefits. Also, such copies of data can be used as backups when the primary copies are damaged or unavailable.

Another function of the grid is to better balance resource utilization. An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid enabled, they can be moved to underutilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and many other kinds of resources that may be available on a grid. Management can use a grid to better view the usage patterns in the larger organization, permitting better planning when upgrading systems, increasing capacity, or retiring computing resources no longer needed.

Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive grid application can be thought of as many smaller “subjobs,” each executing on a different machine in the grid. To the extent that these subjobs do not need to communicate with each other, the more “scalable” the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors.

Barriers often exist to perfect scalability. The first barrier depends on the algorithms used for splitting the application among many CPUs. If the algorithm can only be split into a limited number of independently running parts, then that forms a scalability barrier. The second

barrier appears if the parts are not completely independent; this can cause contention, which can limit scalability. For example, if all of the subjobs need to read and write from one common file or database, the access limits of that file or database will become the limiting factor in the application's scalability. Other sources of inter-job contention in a parallel grid application include message communications latencies among the jobs, network communication capacities, synchronization protocols, input-output bandwidth to devices and storage devices, and latencies interfering with real-time requirements.

Applications

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires top mathematics and programming talents, if it is even possible in a given situation. New computation intensive applications written today are being designed for parallel execution and these will be easily grid enabled, if they do not already follow emerging grid protocols and standards.

Virtual resources and virtual organizations for collaboration

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources, as illustrated in Figure 1 on page 4. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid.

Sharing starts with data in the form of files or databases. A "data grid" can expand data capabilities in several ways. First, files or databases can seamlessly span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques.

Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are "virtualized" to give them a more uniform interoperability among heterogeneous grid participants.

The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

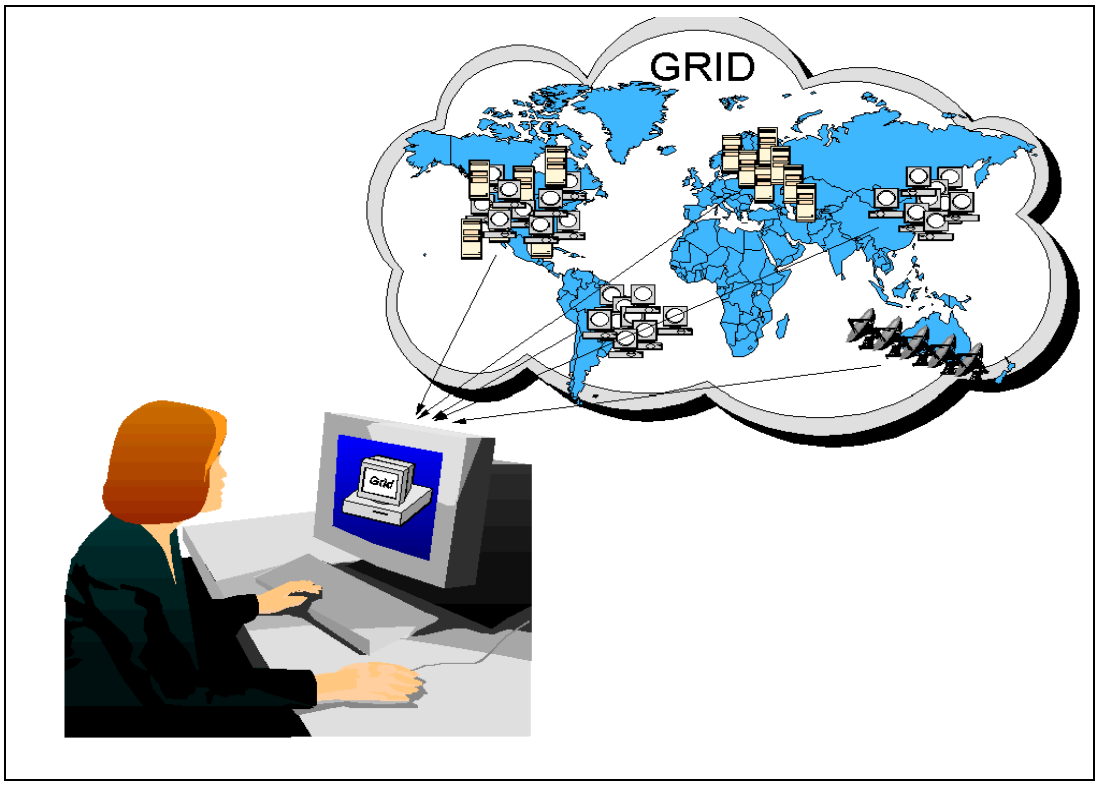

Figure 1 The Grid virtualizes heterogeneous and geographically disperse resources for each virtual organization presenting a simpler view

Access to additional resources

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity.

For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet. In this way, the total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth.

Some machines may have expensive licensed software installed that the user requires. His jobs can be sent to such machines more fully exploiting the software licenses.

Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced color capabilities or faster speeds. Similarly, a grid can be used to make use of other special equipment. For example, a machine may have a high speed, self feeding, DVD writer that could be used to publish a quantity of data faster. Some machines on the grid may be connected to scanning electron microscopes that can be operated remotely. In this case, scheduling and reservation are important. A specimen could be sent in advance to the facility hosting the microscope. Then the user can remotely operate the machine, changing perspective views until the desired image is captured.

The grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination. Today, we have remote device drivers for printers. Eventually, we will

see standards for grid enabled device drivers to many unusual devices and resources. All of these will make the grid look like a large virtual machine with a collection of virtual resources beyond what would be available on just one conventional machine.

Resource balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization, as illustrated in Figure 2. This feature can prove invaluable for handling occasional peak loads of activity in parts of an larger organization. This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the grid.
- If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

Without a grid infrastructure, such balancing decisions are difficult to prioritize and execute.

Occasionally, a project may suddenly rise in importance with a specific deadline. A grid cannot perform a miracle and achieve a deadline when it is already too close. However, if the size of the job is known, if it is a kind of job that can be sufficiently split into subjobs, and if enough resources are available after preempting lower priority work, a grid can bring a very large amount of processing power to solve the problem. In such situations, a grid can, with some planning, succeed in meeting a surprise deadline.

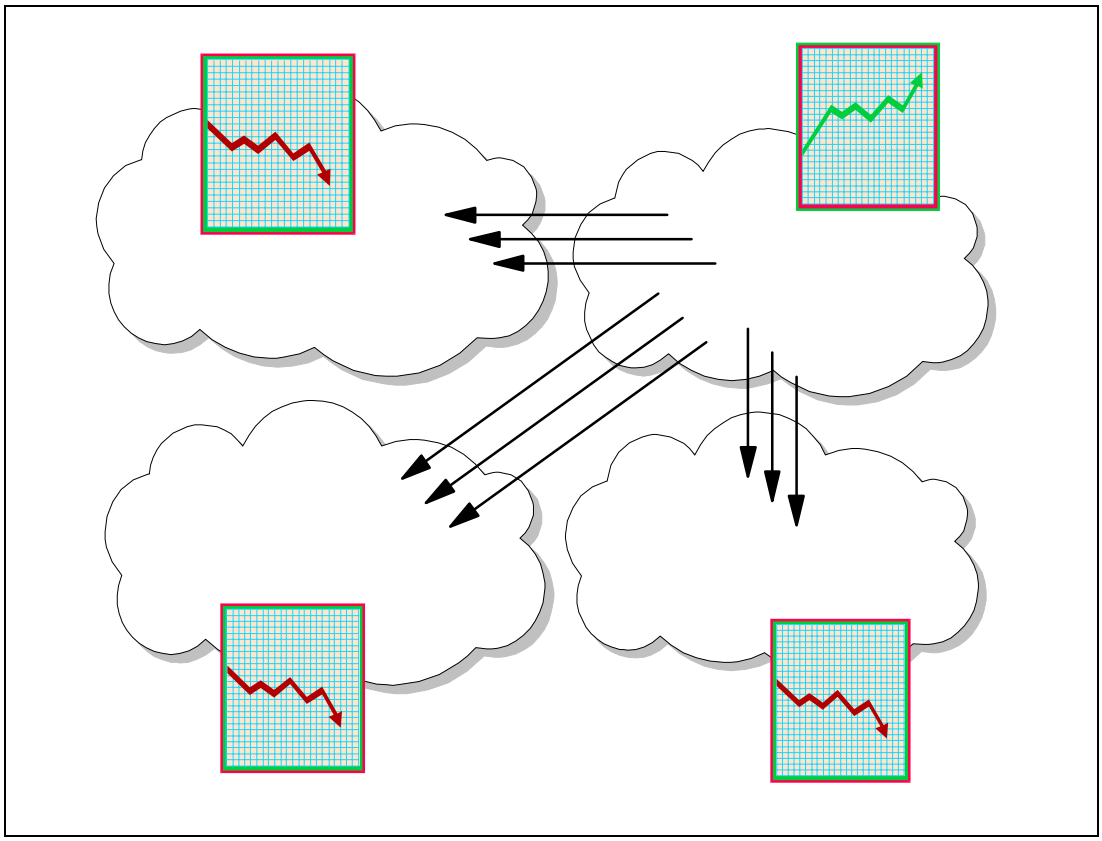

Figure 2 Jobs are migrated to less busy parts of the grid to balance resource loads and absorb unexpected peaks of activity in a part of an organization

Other more subtle benefits can occur using a grid for load balancing. When jobs communicate with each other, the Internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. This can potentially reduce communication and other forms of contention in the grid.

Finally, a grid provides excellent infrastructure for brokering resources. Individual resources can be profiled to determine their availability and their capacity, and this can be factored into scheduling on the grid. Different organizations participating in the grid can build up grid credits and use them at times when they need additional resources. This can form the basis for grid accounting and the ability to more fairly distribute work on the grid.

Reliability

High-end conventional computing systems use expensive hardware to increase reliability. They are built using chips with redundant circuits that vote on results, and contain much logic to achieve graceful recovery from an assortment of hardware failures. The machines also use duplicate processors with hot pluggability so that when they fail, one can be replaced without turning the other off. Power supplies and cooling systems are duplicated. The systems are operated on special power sources that can start generators if utility power is interrupted. All of this builds a reliable system, but at a great cost, due to the duplication of high-reliability components.

In the future, we will see an alternate approach to reliability that relies more on software technology than expensive hardware. A grid is just the beginning of such technology. The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid, as illustrated in Figure 3 on page 7. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering.

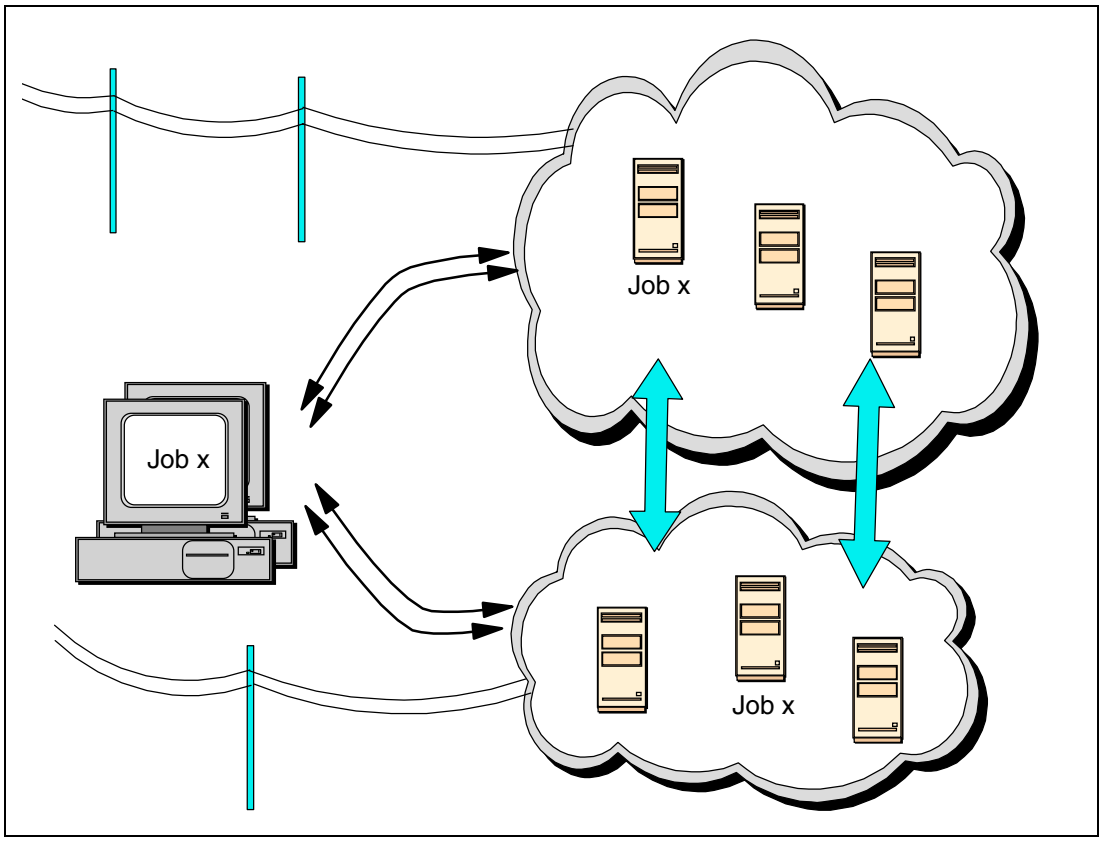

Figure 3 Redundant grid configuration and redundant job submission used to achieve high reliability

Such grid systems will utilize “autonomic computing.” This is a type of software that automatically heals problems in the grid, perhaps even before an operator or manager is aware of them. In principle, most of the reliability attributes achieved using hardware in today’s high availability systems can be achieved using software in a grid setting in the future.

Management

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization.

The grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resource hardware and the expenses associated with it. Often this hardware might be underutilized while another project finds itself in trouble, needing more resources due to unexpected events. With the larger view a grid can offer, it becomes easier to control and manage such situations. As illustrated in Figure 4 on page 8, administrators can change any number of policies that affect how the different organizations might share or compete for resources.

Aggregating utilization data over a larger set of projects can enhance an organization’s ability to project future upgrade needs. When maintenance is required, grid work can be rerouted to other machines without crippling the projects involved.

Autonomic computing can come into play here too. Various tools may be able to identify important trends throughout the grid, informing management of those that require attention.

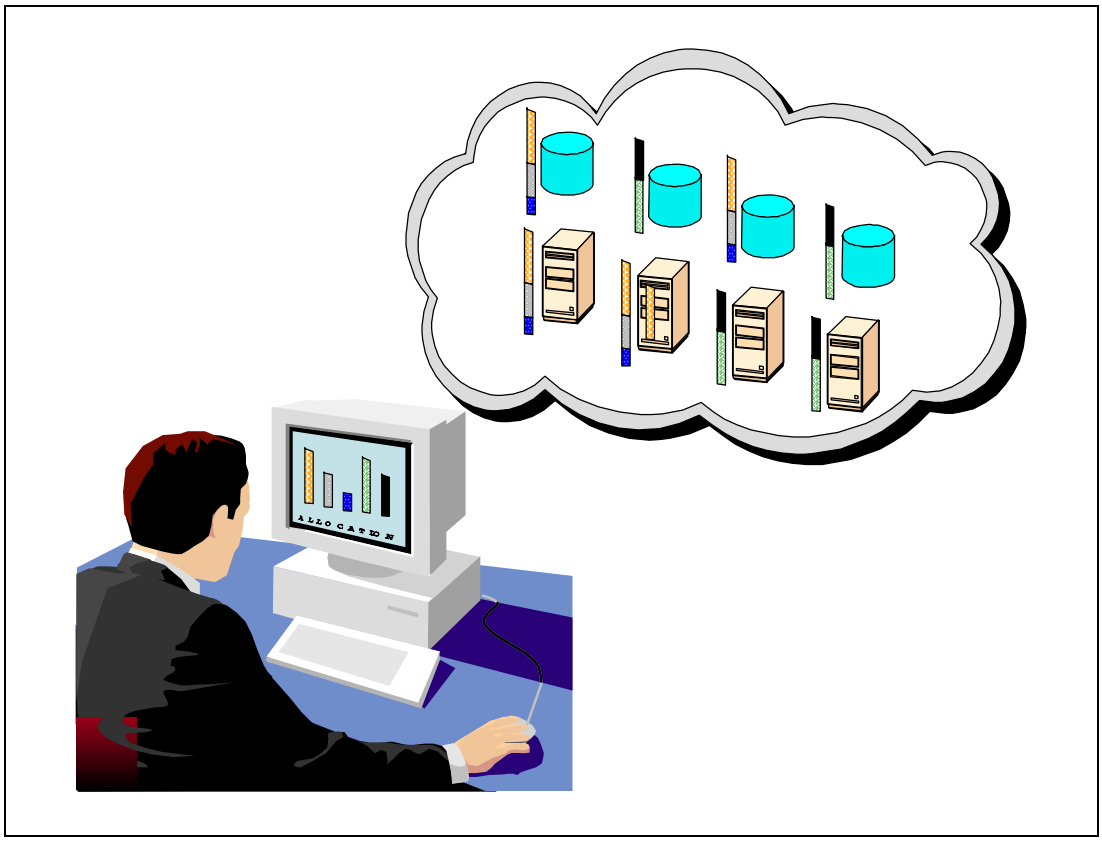

Figure 4 Administrators can adjust policies to better allocate resources

Grid concepts and components

In this section, we introduce the various grid concepts, components, and terms in more detail.

Types of resources

A grid is a collection of machines, sometimes referred to as “nodes,” “resources,” “members,” “donors,” “clients,” “hosts,” “engines,” and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid while others may have specific restrictions.

Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid. The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application that needs to be executed many times on many different machines in the grid. “Scalability” is a measure of how efficiently the multiple processors on a grid are used. If twice as many processors makes an application complete in one half the time, then it is said to be perfectly scalable. However, there may be limits to scalability when applications can only be split into a limited number of

separately running parts or if those parts experience some other contention for resources of some kind.

Storage

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a “data grid.” Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be “secondary storage” using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data to serve as temporary storage for running applications.

Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data. Many grid systems use mountable networked file systems, such as Andrew File System (AFS), Network File System (NFS), Distributed File System (DFS), or General Parallel File System (GPFS). These offer varying degrees of performance, security features, and reliability features.

Capacity can be increased by using the storage on multiple machines with a unifying file system. Any individual file or data base can span several storage devices and machines, eliminating maximum size restrictions often imposed by file systems shipped with operating systems. A unifying file system can also provide a single uniform name space for grid storage. This makes it easier for users to reference data residing in the grid, without regard for its exact location. In a similar way, special data base software can “federate” an assortment of individual data bases and files to form a larger, more comprehensive data base, accessible using data base query functions.

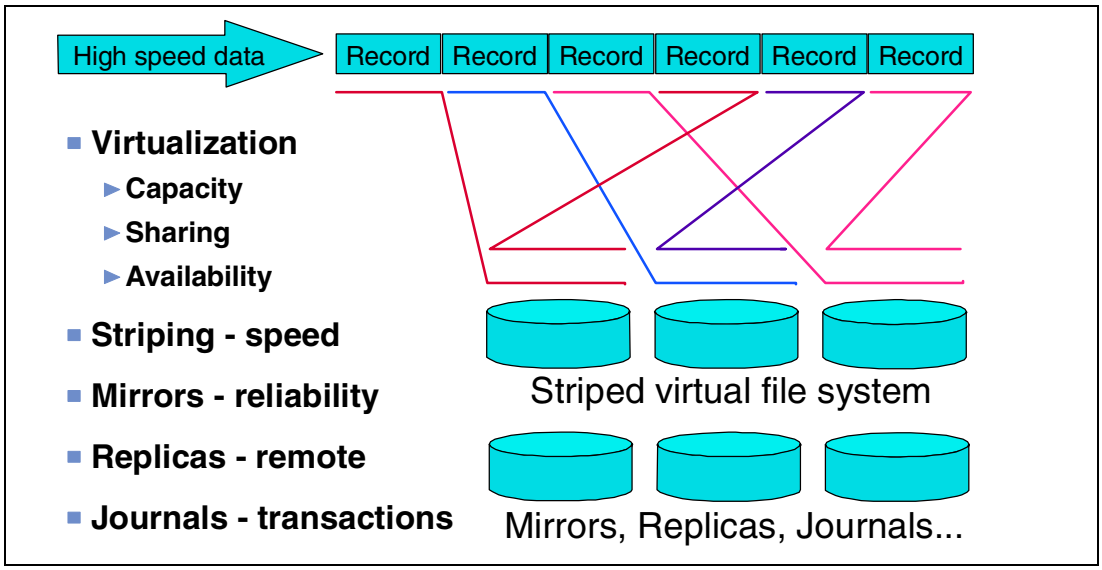

Figure 5 Data striping is writing or reading successive records to/from different physical devices, overlapping the access for faster throughput; additional techniques increase reliability

More advanced file systems on a grid can automatically duplicate sets of data, to provide redundancy for increased reliability and increased performance. An intelligent grid scheduler can help select the appropriate storage devices to hold data, based on usage patterns. Jobs can then be scheduled closer to the data, preferably on the machines directly connected to the storage devices holding the required data.

Data striping can also be implemented by grid file systems, as illustrated in Figure 5 on page 9. When there are sequential or predictable access patterns to data, this technique can create the virtual effect of having storage devices that can transfer data at a faster rate than any individual disk drive. This can be important for multimedia data streams or when collecting large quantities of data at extremely high rates from CAT scans or particle physics experiments, for example.

A grid file system can also implement journaling so that data can be recovered more reliably after certain kinds of failures. In addition, some file systems implement advanced synchronization mechanisms to reduce contention when data is shared and updated by many users.

Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines. When these connections do not share the same communication path, then they add to the total available bandwidth for accessing the Internet.

Redundant communication paths are sometimes needed to better handle potential network failures and excessive data traffic. In some cases, higher speed networks must be provided to meet the demands of jobs transferring larger amounts of data. A grid management system can better show the topology of the grid and highlight the communication bottlenecks. This information can in turn be used to plan for hardware upgrades.

Software and licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed. When the licensing fees are significant, this approach can save significant expenses for an organization.

Some software licensing arrangements permit the software to be installed on all of the machines of a grid but may limit the number of installations that can be simultaneously used at any given instant. License management software keeps track of how many concurrent copies of the software are being used and prevents more than that number from executing at any given time. The grid job schedulers can be configured to take software licenses into account, optionally balancing them against other priorities or policies.

Special equipment, capacities, architectures, and policies

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid.

In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research. These would be identified as having a medical research attribute and the scheduler could be configured to only assign jobs that require machines of the medical research “resource.” Others may be participate in the grid only if they are not used for military purposes. In this situation, jobs requiring a “military resource” would not be assigned to such machines. Of course, the administrators would need to impose a classification on each kind of job through some certification procedure to use this kind of approach.

Jobs and applications

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing “application” or “job.” Usually we use the term “application” as the highest level of a piece of work on the grid. However, sometimes the term “job” is used equivalently. Applications may be broken down into any number of individual jobs, as illustrated in Figure 6. Those, in turn, can be further broken down into “subjobs.” The grid industry uses other terms, such as transaction, work unit, or submission, to mean the same thing as a job.

Jobs are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data, or operate machinery. A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid.

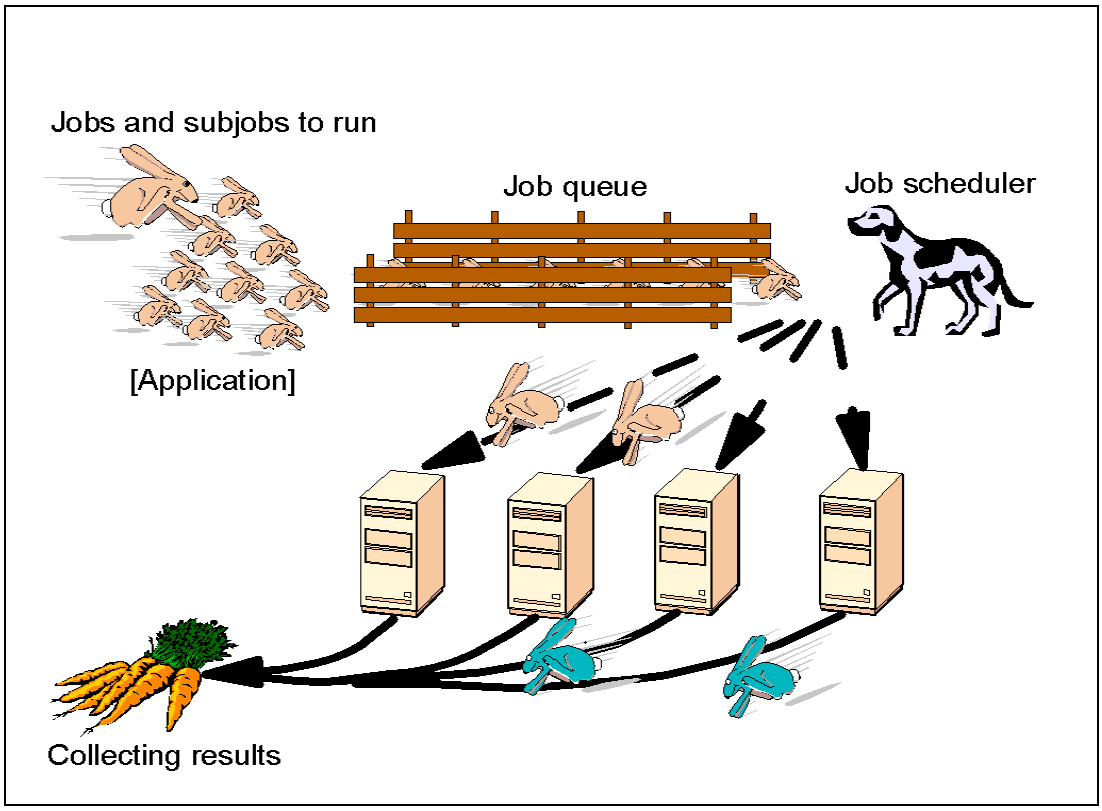

Figure 6 An application is one or more jobs that are scheduled to run on machines in the grid; the results are collected and assembled to produce the answer

The jobs may have specific dependencies that may prevent them from executing in parallel in all cases. For example, they may require some specific input data that must be copied to the machine on which the job is to run. Some jobs may require the output produced by certain

other jobs and cannot be executed until those prerequisite jobs have completed executing. Jobs may spawn additional subjobs, depending on the data they process. This work flow can create a hierarchy of jobs and subjobs. Finally, the results of all of the jobs must be collected and appropriately assembled to produce the ultimate answer for the application.

Scheduling, reservation, and scavenging

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine. More advanced grid systems would include a job “scheduler” of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term “scheduling” is not to be confused with “reservation” of resources in advance to improve the quality of service. Sometimes the term “resource broker” is used in place of “scheduler,” but this term implies that some sort of bartering capability is factored into scheduling.

In a “scavenging” grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job that is satisfied by the machine’s resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid.

To create more predictable behavior, grid machines are often “dedicated” to the grid and are not preempted by outside work. This enables schedulers to compute the approximate completion time for a set of jobs, when their running characteristics are known.

As a further step, grid resources can be “reserved” in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. This is done to meet deadlines and guarantee quality of service. When policies permit, resources reserved in advance could also be scavenged to run lower priority jobs when they are not busy during a reservation period, yielding to jobs for which they are reserved. Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid.

Scheduling and reservation is fairly straightforward when only one resource type, usually CPU, is involved. However, additional grid optimizations can be achieved by considering more resources in the scheduling and reservation process. For example, it would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits. Optimal scheduling, considering multiple resources, is a difficult mathematics problem. Therefore, such schedulers may use heuristics. These heuristics are rules that are designed to improve the probability of finding the best combination of job schedules and reservations to optimize throughput or any other metric.

Intragrid to Intergrid

There have been attempts to formulate a precise definition for what a “grid” is. In fact, the concept of grid computing is still evolving and most attempts to define it precisely end up excluding implementations that many would consider to be grids. We will be pragmatic and not claim to make any definitive descriptions of what a grid is and is not. Therefore, the following descriptions of various kinds of “grids” must be taken loosely.

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world. In this section, we will describe some examples in this range of grid system topologies.

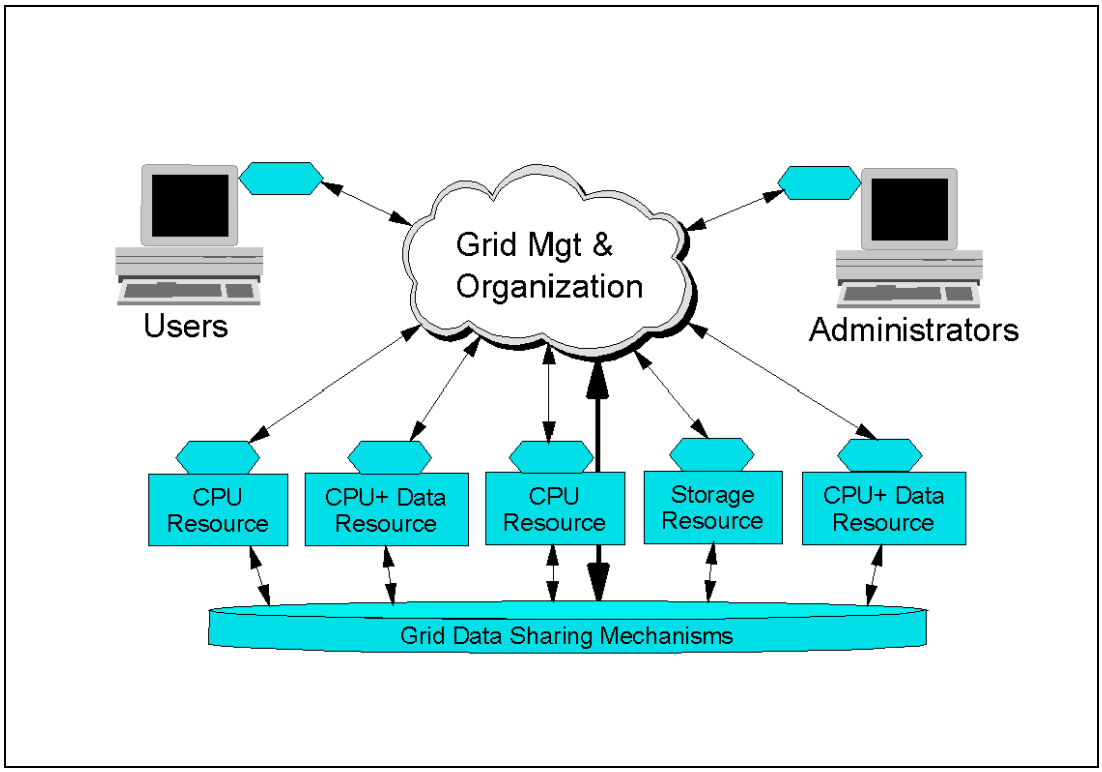

Figure 7 A simple grid

As presented in Figure 7, the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a “cluster” implementation rather than a “grid.”

The next progression would be to include heterogeneous machines. In this configuration, more types of resources are available. The grid system is likely to include some scheduling components. File sharing may still be accomplished using networked file systems. Machines participating in the grid may include ones from multiple departments but within the same organization. Such a grid is also referred to as an “Intragrid.”

As the grid expands to many departments, policies may be required for how the grid should be used. For example, there may be policies for what kinds of work is allowed on the grid and at what times. There may be a prioritization by department or by kinds of applications that should have access to grid resources. Also, security becomes more important as more organizations are involved. Sensitive data in one department may need to be protected from access by jobs running for other departments. Dedicated grid machines may be added to increase the quality of service for grid computing, rather than depending entirely on scavenged resources.

The grid may grow geographically in an organization that has facilities in different cities. Dedicated communications' connections may be used among these facilities and the grid. In some cases, VPN tunneling or other technologies may be used over the Internet to connect the different parts of the organization. Security increases in importance once the bounds of any given facility are traversed. The grid may grow to be hierarchically organized to reduce the contention implied by central control, increasing scalability.

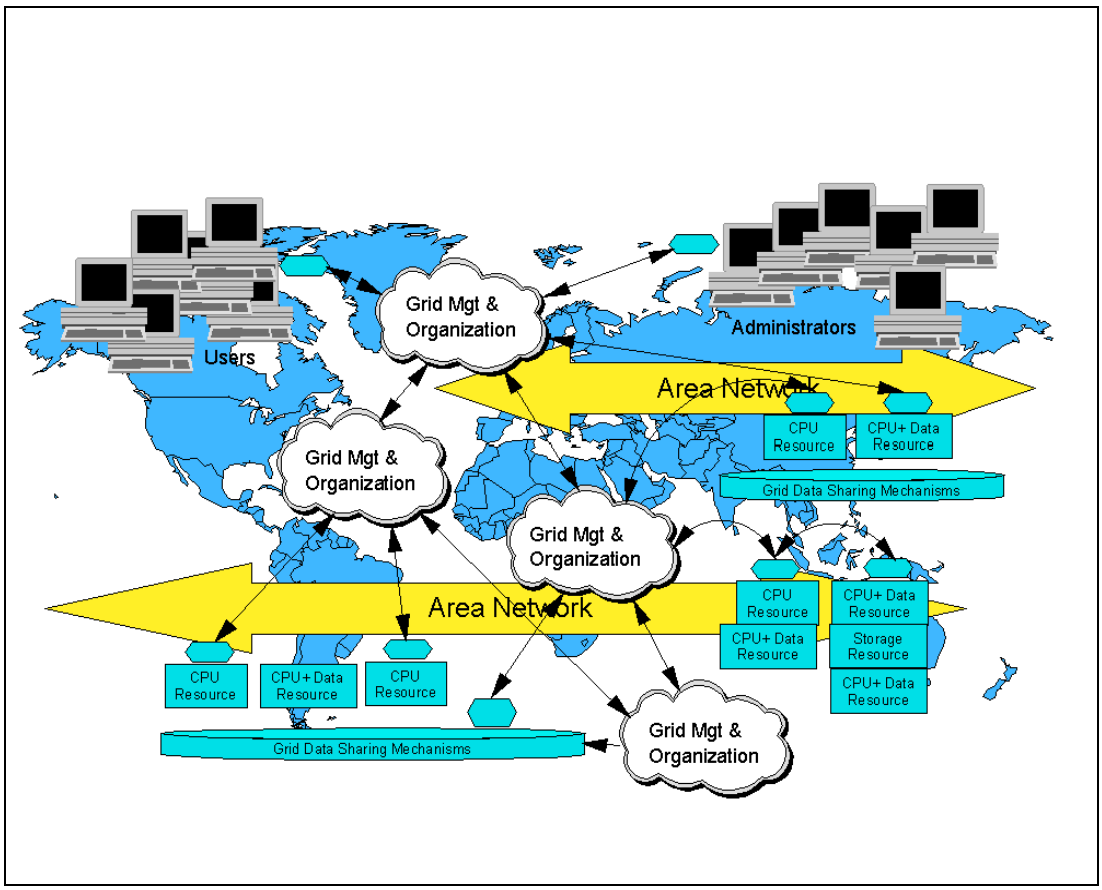

Figure 8 A more complex Intergrid

Over time, as illustrated in Figure 8, a grid may grow to cross organization boundaries, and may be used to collaborate on projects of common interest. This is known as an "Intergrid." The highest levels of security are usually required in this configuration to prevent possible attacks and spying. The Intragrid offers the prospect for trading or brokering resources over a much wider audience. Resources may be purchased as a utility from trusted suppliers.

Grid construction

An *ad hoc* grid may be installed by a few programmers in their spare time, but as the grid grows, and as users become more dependent on it for mission-critical work, a degree of planning is essential. It is best to understand the organization's requirements and choose grid technologies that best fit these requirements. This section discussed some of the planning considerations and grid components that address the requirements.

Deployment planning

The use of a grid is often born from a need for increased resources of some type. One often looks to their neighbor who may have excess capacity in the particular resource. One of the first considerations is the hardware available and how it is connected via a LAN or WAN. Next, an organization may want to add additional hardware to augment the capabilities of the grid. It is important to understand the applications to be used on the grid. Their characteristics can affect the decisions of how to best choose and configure the hardware and its data connectivity.

Security

Security is a much more important factor in planning and maintaining a grid than in conventional distributed computing, where data sharing comprises the bulk of the activity. In a grid, the member machines are configured to execute programs rather than just move data. This makes an unsecured grid potentially fertile ground for viruses and Trojan horse programs. For this reason, it is important to understand exactly which components of the grid must be rigorously secured to deter any kind of attack. Furthermore, it is important to understand the issues involved in authenticating users and properly executing the responsibilities of a certificate authority.

Organization

The technology considerations are important in deploying a grid. However, organizational and business issues can be equally important. It is important to understand how the departments in an organization interact, operate, and contribute to the whole. Often, there are barriers built between departments and projects to protect their resources in an effort to increase the probability of timely success. However, by rethinking some of these relationships, one can find that more sharing of resources can sometimes benefit the entire organization better. For example, a project that finds itself behind schedule and over budget may not be able to afford the resources required to solve the problem. A grid would give such projects an added measure of safety, providing an extra margin of resource capacity needed to finish the project. Similarly, a project in its early stages, when computing resources are not being fully utilized, may be able to donate them to other projects in need. A grid also offers the ability for the organization's management to see the bigger priority picture and react more quickly in shifting resource utilization, priorities, and policies.

Grid software components

This section presents some of the key components that must be discussed before designing a grid computing architecture.

Management components

Any grid system has some management components. First, there is a component that keeps track of the resources available to the grid and which users are members of the grid. This information is used primarily to decide where grid jobs should be assigned.

Second, there are measurement components that determine both the capacities of the nodes on the grid and their current utilization rate at any given time. This information is used to schedule jobs in the grid. Such information is also used to determine the health of the grid, alerting personnel to problems such as outages, congestion, or overcommitment. This information is also used to determine overall usage patterns and statistics, as well as to log and account for usage of grid resources.

Third, advanced grid management software can automatically manage many aspects of the grid. This is known as "autonomic computing," or "recovery oriented computing." This

software would automatically recover from various kinds of grid failures and outages, finding alternative ways to get the workload processed.

Donor software

Each machine contributing resources typically needs to enroll as a member of the grid and install some software that manages the grid's use of its resources. Usually, some sort of identification and authentication procedure must be performed before a machine can join the grid. A certificate authority can be used to establish the identity of the donor machine as well as the users and the grid itself.

Some grid systems provide their own login to the grid while others depend on the native operating systems for user authentication. In the latter case, a user ID mapping system may be needed to match the user's rights properly on different machines. This typically is manually maintained by a grid administrator. He determines which user ID a given user may possess on each grid machine and enters these IDs in a protected data base or registry. In this way, when grid jobs are submitted to different machines for a user, the proper local machine user ID is used for determining the users rights.

In some grid systems, it is possible to join the grid without any special authentication. And in others, it is possible for any user to submit jobs to the grid. Such systems may be convenient to set up, but should be discouraged in larger deployments due to the serious security problems that they would open up.

The grid system makes information about the newly added resources available throughout the grid. The donor machine will usually have some sort of monitor that determines or measures how busy the machine is and the rate or amount of resources utilized. This information is "bubbled up" to the management software of the grid and used to schedule use of those resources accordingly. In a scavenging system, this information tells the grid management software when the machine is idle and available for work.

Most importantly, the software installed on a given machine can accept an executable job from the grid management system and execute it. A user somewhere on the grid submits a job for execution on the grid. The grid management software must communicate with the grid donor software to send the job there. The donor grid software must be able to receive the executable file or select the proper one from copies pre-installed on the donor machine. The software is executed and the output is sent back to the requester. More advanced implementations can dynamically adjust the priority of a running job, suspend it and resume it later, or checkpoint it with the possibility of resuming its execution on a different machine. These kinds of actions may be necessary to respond to load balancing problems or priority or policy changes in the grid.

Submission software

Usually any member machine of a grid can be used to submit jobs to the grid and initiate grid queries. However, in some grid systems, this function is implemented as a separate component installed on "submission nodes" or "submission clients." When a grid is built using dedicated resources rather than scavenged resources, separate submission software is usually installed on the user's desktop or workstation.

Distributed grid management

Larger grids may have a hierarchical or other type of organizational topology usually matching the connectivity topology. That is, machines locally connected together with a LAN form a "cluster" of machines. The grid may be organized in a hierarchy consisting of clusters of clusters. The work involved in managing the grid is distributed to increase the scalability of the grid. The collection and grid operation and resource data as well as job scheduling is distributed to match the topology of the grid. For example, a central job scheduler will not

schedule a submitted job directly to the machine which is to execute it. Instead the job is sent to a lower level scheduler which handles a set of machines (or further clusters). The lower level scheduler handles the assignment to the specific machine. Similarly, the collection of statistical information is distributed. Lower level clusters receive activity information from the individual machines, aggregate it, and send it to higher level management nodes in the hierarchy.

Schedulers

Most grid systems include some sort of job scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler.

Some schedulers implement a job priority system. This is sometimes done by using several job queues, each with a different priority. As grid machines become available to execute jobs, the jobs are taken from the highest priority queues first. Policies of various kinds are also implemented using schedulers. Policies can include various kinds of constrains on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

Schedulers usually react to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower level scheduler rather than to an individual machine.

More advanced schedulers will monitor the progress of scheduled jobs managing the overall work-flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for re-submission and some of which are not.

Reserving resources on the grid in advance is accomplished with a “reservation system.” It is more than a scheduler. It is first a calendar based system for reserving resources for specific time periods and preventing any others from reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached.

Communications

A grid system may include software to help jobs communicate with each other. For example, an application may split itself into a large number of subjobs. Each of these subjobs is a separate job in the grid. However, the application may implement an algorithm that requires that the subjobs communicate some information among them. The subjobs need to be able to locate other specific subjobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) and any of several variations is often included as part of the grid system for just this kind of communication.

Observation, management, and measurement

We mentioned above the schedulers react to current loads on the grid. Usually, the donor software will include some tools that measure the current load and activity on a given machine using either operating system facilities or by direct measurement. This software is sometimes referred to as a “load sensor.” Some grid systems provide the means for implementing custom load sensors for other than CPU or storage resources.

Such measurement information is useful not only for scheduling, but also for discovering overall usage patterns in the grid. The statistics can show trends which may signal the need for additional hardware. Also, measurement information about specific jobs can be collected and used to better predict the resource requirements of that job the next time it is run. The better the prediction, the more efficiently the grid's workload can be managed.

The measurement information can also be saved for accounting purposes, to form the basis for grid resource brokering, or to manage priorities more fairly. The information can also be displayed in various forms to better visualize grid activity and utilization.

Using a grid: A user's perspective

This section describes the typical usage activities in using the grid from an user's perspective.

Enrolling and installing grid software

A user first enrolls as a grid user, and installs the provided grid software on his own machine. He may optionally enroll his machine as a donor on the grid.

Enrolling in the grid may require authentication for security purposes. The user positively establishes his identity with a certificate authority. This should not be done solely via the Internet. The certificate authority must take steps to assure that the user is in fact who he claims to be. The certificate authority makes a special certificate available to software needing to check the true identity of a grid user and his grid requests. Similar steps may be required to identify the donating machine. The user has the responsibility of keeping his grid credentials secure.

Once the user and/or machine are authenticated, the grid software is provided to the user for installing on his machine for the purposes of using the grid as well as donating to the grid. This software may be automatically preconfigured by the grid management system to know the communication address of the management nodes in the grid and user or machine identification information. In this way, the installation may be a one click operation with a minimum of interaction required on the part of the user. In less automated grid installations, the user may be asked to identify the grid's management node and possibly other configuration information. He may choose to limit the resources donated to the grid, the times that his machine is usable by the grid, and other policy related constraints. The user may also need to inform the grid administrator which user IDs are his on other machines that exist on the grid.

Logging onto the grid

To use the grid, most grid systems require the user to log on to a system using a user ID that is enrolled in the grid. Other grid systems may have their own grid login ID separate from the one on the operating system. A grid login is usually more convenient for grid users. It eliminates the ID matching problems among different machines. To the user, it makes the grid look more like one large virtual computer rather than a collection of individual machines. Globus, for example, implements a proxy login model that keeps the user logged in for a specified amount of time, even if he logs off and back on the operating system and even if the machine is rebooted.

Once logged on, the user can query the grid and submit jobs. Some grid implementations permit some query functions if the user is not logged into the grid or even if the user is not enrolled in the grid.

Queries and submitting jobs

The user will usually perform some queries to check to see how busy the grid is, to see how his submitted jobs are progressing, and to look for resources on the grid. Grid systems usually provide command line tools as well as graphical user interfaces (GUIs) for queries. Command line tools are especially useful when the user wants to write a script that automates a sequence of actions. For example, the user might write a script to look for an available resource, submit a job to it, watch the progress of the job, and present the results when the job has finished.

Job submission usually consists of three parts, even if there is only one command required. First, some input data and possibly the executable program or execution script file are sent to the machine to execute the job. Sending the input is called “staging the input data.” Alternatively, the data and program files may be pre-installed on the grid machines or accessible via a mountable networked file system. When the grid consists of heterogeneous machines, there may be multiple executable program files, each compiled for the different machine platforms on the grid. A nice feature provided by some grid systems is to register these multiple versions of the program so that the grid system can automatically choose a correctly matching version to the grid machine that will run the program. Some grid technologies require that the program and input data be first processed or “wrappered” in some way by the grid system. This may be done to add protective execution controls around the application or just to simply collect all of the data files into one.

Second, the job is executed on the grid machine. The grid software running on the donating machine executes the program in a process on the user’s behalf. It may use a common user ID on the machine or it may use the user’s own user ID, depending on which grid technology is used. Some grid systems implement a protective “sandbox” around the program so that it cannot cause any disruption to the donating machine if it encounters a problem during execution. Rights to access files and other resources on the grid machine may be restricted.

Third, the results of the job are sent back to the submitter. In some implementations, intermediate results can be viewed by the user who submitted the job. In some grid technologies that do not automatically stage the output data back to the user, the results must be explicitly sent to the user, perhaps using a networked file system.

Scripts are also useful for submitting a series of jobs, for a parameter space application, for example. Some computation problems consist of a search for the desired result based on some input parameters. The goal is to find the input parameters that produce the best desired result. For each input parameter, a separate job is executed to find the result for that value. The whole application consists of many such jobs, which explore the results for a large number of input parameter values. Scripts are usually used to launch the many subjobs, each receiving their own particular parameter values. Parameter inputs can sometimes be more complex than simply a number. Sometimes a different input data set represents the “input parameter.” Scripts help automate the large variety of more complex parameter space study problems. For simpler parameter space inputs, some grid products provide a GUI to submit the series of subjobs, each with different input parameter values.

When there are a large number of subjobs, the work required to collect the results and produce the final result is usually accomplished by a single program, usually running on the machine at the point of job submission. If there are a very large number subjobs required for an application, the work of collecting the results might be distributed as well. For example, the subjob that submits more subjobs to the grid would be responsible for collecting and aggregating the results of the subjobs it spawned.

Data configuration

The data accessed by the grid jobs may simply be staged in and out by the grid system. However, depending on its size and the number of jobs, this can potentially add up to a large amount of data traffic. For this reason, some thought is usually given on how to arrange to have the minimum of such data movement on the grid.

For example, if there will be a very large number of sub-jobs running on most of the grid systems for an application that will be repeatedly run, the data they use may be copied to each machine and reside until the next time the application runs. This is preferable to using a networked file system to share this data, because in such a file system, the data would be effectively moved from a central location every time the application is run. Thus is true unless the file system implements a caching feature or replicates the data automatically.

There are many considerations in efficiently planning the distribution and sharing of data on a grid. This type of analysis is necessary for large jobs to better utilize the grid and not create unnecessary bottlenecks.

Monitoring progress and recovery

The user can query the grid system to see how his application and its subjobs are progressing. When the number of subjobs becomes large, it becomes to difficult to list them all in a graphical window. Instead, there may simply be a one large bar graph showing some averaged progress metric. It becomes more difficult for the user to tell if any particular subjob is not running properly.

A grid system, in conjunction with its job scheduler, often provides some degree of recovery for subjobs that fail. A job may fail due to a:

- Programming error: The job stops part way with some program fault.
- Hardware or power failure: The machine or devices being used stop working in some way.
- Communications interruption: A communication path to the machine has failed or is overloaded with other data traffic.
- Excessive slowness: The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention.

It is not always possible to automatically determine if the reason for a job's failure is due to a problem with the design of the application or if it is due to failures of various kinds in the grid system infrastructure. Schedulers are often designed to categorize job failures in some way and automatically resubmit jobs so that they are likely to succeed, running elsewhere on the grid. In some systems, the user is informed about any job failures and the user must decide whether to issue a command to attempt to rerun the failed jobs.

Grid applications can be designed to automate the monitoring and recovery of their own subjobs using functions provided by the grid system software application programming interfaces (APIs).

Reserving resources

To improve the quality of a service, the user may arrange to reserve a set of resources in advance for his exclusive or high priority use. A calendaring system analogy can be used here. Such a reservation system can also be used in conjunction with planned hardware or software maintenance events, when the affected resource might not be available for grid use.

In a scavenging grid, it may not be possible to reserve specific machines in advance. Instead, the grid management systems may allocate a larger fraction of its capacity for a given reservation to allow for the likelihood of some of the resources becoming unavailable. This must be done in conjunction with tools that have profiled the grid's workload capacity sufficiently to have reliable statistics about the grid's ability to serve the reservation.

Using a grid: An administrator's perspective

This section describes the typical usage activities in using the grid from an administrator's perspective.

Planning

The administrator should understand the organization's requirements for the grid to better choose the grid technologies that satisfy those requirements. The following sections briefly describe the steps the administrator may take to manage the grid. It is suggested that one should start by deploying a small grid first, to learn about its installation and management, before having to confront more complicated issues involved with a large grid.

Installation

First, the selected grid system must be installed on an appropriately configured set of machines. These machines should be connected using networks with sufficient bandwidth to other machines on the grid. Of prime importance is understanding the fail-over scenarios for the given grid system so that the grid can continue operating even if any of the management machines fails in some way. Machines should be configured and connected to facilitate recovery scenarios. Any critical data bases or other data essential for keeping track of the jobs in the grid, members of the grid, and machines on the grid should have suitable backups. Furthermore, public key certificates must be backed up and the private keys must be held in a highly secured place inaccessible by anyone else.

After installation, the grid software may need to be configured for the local network address and IDs. The administrator will usually require root access to the machines managing the grid. In some grid systems, he will also need root access to the donor machines be required to install the software on those as well. The software to be installed on the donor machines may need to be customized so that it can find the grid management machines automatically and include pre-installed public keys for the grid. This software may be provided to potential donors on an FTP or equivalent server or be made available on physical media.

Once, the grid is operational, there may be application software and data that should be installed on donor machines as well. This software may have specific licensing restrictions that should be understood and adhered to. Some grid systems include tools to assist with grid-wide license management. This can both help in following the rules of the licenses and most efficiently exploit those licenses.

Managing enrollment of donors and users

An ongoing task for the grid administrator is to manage the members of the grid, both the machines donating resources and the users. Users may be further organized as project groups. The administrator is responsible for controlling the rights of the users in the grid. Donor machines may have access rights that require management as well. Grid jobs running on donor machines may be executed under a special grid user ID on behalf of the users

submitting the jobs. The rights of these grid user IDs must be properly set so that grid jobs do not allow access to parts of the donor machine to which the users are not entitled.

As users join the grid, their identity must be positively established and entered in the certificate authority. The user and his certificate credentials must be added to the user list using the software appropriate for the grid system deployed. In some cases, the administrator must propagate the user information to several or all grid machines. Also, when the grid system depends primarily on the operating system for user login, the administrator may need to add entries to map the grid user to specific operating system user IDs on the donor machines.

Similar enrollment activity is usually required to enroll donor machines into the grid. The machine's identity is established and registered with the certificate authority. The administrator of the grid must have an agreement with the administrator of the donor machine about user IDs, software, access rights, and any policy restrictions. The administrator must enter the machine's identification credentials, addresses, and resource characteristics using the appropriate software for enrolling the donor machine into the grid. In some cases, the administrator may need to manually propagate this information to other machines in the grid.

Corresponding procedures for removing users and machines must be executed by the administrator.

Certificate authority

It is critical to ensure the highest levels of security in a grid because the grid is designed to execute code and not just share data. Thus, it can be fertile ground for viruses, Trojan horses, and other attacks if the grid system is compromised in any way. The certificate authority is one of the most important aspects of maintaining strong grid security. An organization may choose to use an external certificate authority or operate one itself. You must be able to trust the certificate authority to strictly adhere to its responsibilities.

The primary responsibilities of a certificate authority are:

- Positively identify entities requesting certificates
- Issuing, removing, and archiving certificates
- Protecting the certificate authority server
- Maintaining a namespace of unique names for certificate owners
- Serve signed certificates to those needing to authenticate entities
- Logging activity

Briefly, a certificate authority is based on the public key encryption system. In this system, keys are generated in pairs, a public key and a private key. Either one can be used to encrypt some data such that the other is needed to decrypt it. The private key is guarded by the owner and never revealed to anyone. The public one is given to anyone needing it. A certificate authority is used to hold these public keys and to guarantee who they belong to. When a user uses his private key to encrypt something, the receiver uses the corresponding public key to decrypt it. The receiver knows that only that user's public key can decrypt the message correctly. However, anyone could intercept this message and decrypt it because anyone can get the originator's public key. If the originator instead doubly encrypts the message with his private key and the intended recipient's public key, a secure communication link is formed. The receiver uses his private key to decrypt the message and then uses the sender's public key for the second decryption. Now the recipient knows that if the message decrypts properly, then only the sender could have sent it and furthermore, the sender knows that only the intended receiver can decrypt it. The beauty of all of this is that nobody had to

securely carry an encryption key from the sender to the receiver, as must be done for conventional encryption systems, and any tampering with the communication is revealed. A similar exchange is used to get anyone's public key from the certificate authority, so that the user knows that he has received an unaltered public key for the desired user.

Resource management

Another responsibility of the administrator is to manage the resources of the grid. This includes setting permissions for grid users to use the resources as well as tracking resource usage and implementing a corresponding accounting or billing system. Usage statistics are useful in identifying trends in an organization that may require the acquisition of additional hardware, reduction in excess hardware to reduce costs, and adjustments in priorities and policies to achieve utilization that is fairer or better achieves the overall goals of an organization.

Some grid components, usually job schedulers, have provisions for enforcing priorities and policies of various kinds. It is the responsibility of the administrator to configure these to best meet the goals of the overall organization. Software license managers can be used in a grid setting to control the proper utilization. These may be configured to work with job schedulers to prioritize the use of the limited licenses.

Data sharing

For small grids, the sharing of data can be fairly easy, using existing networked file systems, databases, or standard data transfer protocols. As a grid grows and the users become dependent on any of the data storage repositories, the administrator should consider procedures to maintain backup copies and replicas to improve performance. All of the resource management concerns apply to data on the grid.

Using a grid: An application developer's perspective

Grid applications can be categorized in one of the following three categories:

- Applications that are not enabled for using multiple processors but can be executed on different machines.
- Applications that are already designed to use the multiple processors of a grid setting.
- Applications that need to be modified or rewritten to better exploit a grid.

The latter category is of interest to grid application developers. They will find a need for tools for debugging and measuring the behavior of grid applications. Such grid based tools are still in their infancy. It may be useful for developers to configure a small grid of their own so that they can use debuggers on each machine to control and watch the detailed workings of the applications. Since the debugging process can bypass certain security precautions, it may not always be wise to allow such debugging on a production grid.

Globus is more a developer's toolkit for building grid components rather than a comprehensive grid system. It has the basic components needed to build new facilities to manage grid operations, measurement, repair, and debug grid applications. Tools conforming to the emerging Open Grid Services Architecture (OGSA) interfaces will be usable on various vendor grid systems.

The present and the future

The Globus toolkit is a set of tools useful for building a grid. Its strength is a good security model, with a provision for hierarchically collecting data about the grid, as well as the basic facilities for implementing a simple, yet world-spanning grid. Globus will grow over time through the work of many organizations that are extending its capabilities. More information about Globus can be obtained at http://www.globus.org.

Most grid systems include some job schedulers, but as grids span wider areas, there will be a need for more meta-schedulers that can manage variously configured collections of clusters and smaller grids. These schedulers will evolve to better schedule jobs, considering multiple resources rather than just CPU utilization. They will also extend their reach to implement better quality of service, using reservations, redundancy, and history profiles of jobs and grid performance.

Today, grid systems are still at the early stages of providing a reliable, well performing, and automatically recoverable virtual data sharing and storage. We will see products that take on this task in a grid setting, federating data of all kinds, and achieving better performance, integration with scheduling, reliability, and capacity.

Autonomic computing has the goal to make the administrator's job easier by automating the various complicated tasks involved in managing a grid. These include identifying problems in real time and quickly initiating corrective actions before they seriously impair the grid.

Open Grid Services Architecture (OGSA) is an open standard at the base of all of these future grid enhancements. OGSA will standardize the grid interfaces that will be used by the new schedulers, autonomic computing agents, and any number of other services yet to be developed for the grid. It will make it easier to assemble the best products from various vendors, increasing the overall value of grid computing. More information about OGSA can be obtained at http://www.globus.org/ogsa.

What the grid cannot do

A word of caution should be given to the overly enthusiastic. The grid is not a silver bullet that can take any application and run it a 1000 times faster without the need for buying any more machines or software. Not every application is suitable or enabled for running on a grid. Some kinds of applications simply cannot be parallelized. For others, it can take a large amount of work to modify them to achieve faster throughput. The configuration of a grid can greatly affect the performance, reliability, and security of an organization's computing infrastructure. For all of these reasons, it is important for the us to understand how far the grid has evolved today and which features are coming tomorrow or in the distant future.

Acknowledgements

This publication was produced during the grid computing project, organized and coordinated by the International Technical Support Organization in October 2002. The following team of specialists, from around the world, contributed to this project:

Jonathan Armstrong and Viktors Berstis
Grid Computing Initiative, e-Technology Center - IBM Austin

Mike Kendzierski
ITS Northeast Region - IBM North America

Andreas Neukoetter
iBM @server OGSA Development - IBM Germany

Richard Bing-Wo
Professional Services Consultant - IBM North America

Olegario Hernandez
Business Partner - IBM Chile

Masanobu Takagi
IBM Japan

Adeeb Amir
Professional Services Consultant - IBM North America

Ryo Murakawa
IBM Japan

Norbert Bieberstein
Solution Development Manager - IBM Germany

Luis Ferreira
Grid and Linux Team, International Technical Support Organization - IBM Austin

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright IBM Corp. 2002. All rights reserved.

This document created or updated on November 11, 2002.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. JN9B Building 003 Internal Zip 2834
  11400 Burnet Road
  Austin, Texas 78758-3493 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS®	DFS™	PowerPC®
IBM®	Redbooks™	
IBM eServer™	Redbooks(logo)™ 	

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Globus Project™ and Globus Toolkit™ are trademarks held by the University of Chicago.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.