



GemFire Enterprise Data Fabric (EDF) for High Performance Grid Computing

GRID COMPUTING: THE CHANGING LANDSCAPE

Grids computing is often viewed as a vehicle to realize the promise of distributed computing in large-scale heterogeneous environments. Simply put, it enables the virtualization of key resources like CPU, memory, disk and storage spread across disparate systems as a single managed entity. To grid consumers, it appears as though they are dealing with a giant supercomputer with almost infinite memory, CPU and disk space.

Grid computing solutions are often tasked with the following basic functional requirements:

- Managing and virtualizing a distributed set of resources.
- Scheduling tasks (in parallel) across a set of compute nodes.
- Maintaining a desired level of quality of service.

The benefits that stem from successfully addressing the above requirements are multi-fold. First, grids engender a cost-effective computing paradigm that does not necessitate specialized or expensive hardware and software. Secondly, optimal utilization of resources is guaranteed. Unused compute power and disk space is made available to processes that can utilize them. Individual resources can be profiled to determine their availability and their capacity, and this information can be factored into scheduling on the grid. Thirdly, a grid can serve as a mechanism to provision and mobilize resources to scale to sporadic or unexpected loads. This is in some senses similar to how an electric power grid works to meet the varying demands of the consumers in its grid. Similarly, if a particular marketing promotion is delivering more than expected hits on a shopping web-site, a grid can scale accordingly to accommodate user transactions. Grids can also facilitate collaboration across applications and enable sharing of data once the necessary data infrastructure is in place. Lastly, overall operational reliability and quality of service can be guaranteed for applications deployed on a grid. In this day and age of stringent Service Level Agreements (SLA), such a guarantee becomes a strong business advantage.

A detailed analysis of grid applications reveals that such applications can be divided into two categories based on their function - long-running process grids and high-performance real-time grids. The first category involves grids used in areas such as overnight Value-at-Risk (VaR) calculations in investment bank, or life sciences research, or the SETI project, or in the area of computational fluid dynamics (CFD). Complex numerical processes that run for long periods of time characterize these grid deployments. The results from one set of runs usually feed another set of runs and the data transfer is handled through FTP or batch replication schemes. On the other end of spectrum are the high-performance real-time grids for applications such as complex pricing or client-exposure/risk modeling in capital markets or risk management in insurance, or fraud detection in banking and telecom. The need for low-latency, scalability and high availability in these applications is absolutely critical. Such grids are deployed to meet the

real-time needs of an enterprise and are often required to scale out to meet growing demands. For instance, investment banks have an insatiable appetite when it comes to computations and analytics. Hence, the number of grid nodes deployed for such purposes keeps growing continually.

Most recent popular trend is the synergy between Service Oriented Architectures (SOA) and grids. Grids are being considered as a deployment topology for SOA. The applications deployed on a grid are exposed in SOA as Web Services and virtualized to clients through a services layer. As discussed before, grid applications themselves have a virtualization layer between themselves and the actual compute, memory and storage resources. The net result is a cascading virtualization pattern, which consists of a service virtualization layer between the users and grid/composite applications, backed by resource virtualization layer between the grids and the actual physical systems.

THE LOOMING CHALLENGE

Given the adoption of grid computing across multiple industries over the last couple of years, a few technology trends that impact as well as challenge this adoption merit discussion. First, the 'need for speed' or improved operational efficiency is becoming a prevalent trend in multiple industries like capital markets, insurance, retail and healthcare. This requirement obviously places an extra burden on grid deployments and challenges their success. Whether it is faster pricing of securities or quicker actuarial data processing or claims processing, latency impacts profitability, customer service, and risk management. Secondly, the data volumes that need to be managed on a grid are also increasing. For instance, the growing popularity of electronic trading in capital markets has a concomitant effect on the data volumes that brokerage firms need to consider for computations and analytics. This trend again impacts the scalability of a grid. Lastly, the popular adoption of distributed hardware architectures like blades is a double-edged sword when it comes to grid deployments. On the one hand, it greatly reduces the efforts involved in the horizontal scaling of a grid - thanks to the plug-and-play model of blade servers. But on the other hand, these distributed hardware deployments necessitate a high-speed data distribution mechanism that guarantees data on demand on every node to suitably justify the hardware investment.

Given these trends, it is no surprise that the Grid Adoption Research Services (GARS) report by the 451 Group revealed that 'many large enterprises have delayed broadening their grid deployments because of limitations in data management capabilities'.

The limitations indicated in the aforementioned report can be filtered down to the following five main data related issues that often cripple grid deployments:

- **High data access latency** - Grid applications typically spend too much time waiting on the data required for computations. This is typically due to disk-bound data access and transformations on data from databases or file-systems.
- **Lack of scalability to increasing data volumes** - Most grid deployments lack the ability to handle large data volumes due to lack of intelligent data partitioning schemes and low latency data distribution mechanisms that can support an increase in the number of grid nodes.
- **Inability to share and distribute data across nodes** - This is a usual problem in grids that have multiple processes working on a shared piece of data that is constantly being updated, or in scenarios where the results from one grid process need to be shared as inputs for other processes instantaneously.
- **Data consistency issues** - In certain deployments, grid applications maintain local data copies for faster access. This works fine as long as the data is read-only. But, such a model spells disaster if the data is changing. The net result is multiple inconsistent data copies used by different grid processes.
- **Reliability and Quality of Service problems** - If an application on a particular grid node fails, it is usually accompanied by data loss. Such a loss could easily cause other grid operations to fail as well and cause downtime and consequent reliability issues.

So, what is the root cause of these problems? Today, the word 'grid' often refers to compute grids with no explicit reference to data semantics. Data management is typically an afterthought in most grid deployments. This needs to change and such deployments require a new look at data architecture and data grid patterns. Traditional RDBMS, file-systems, messaging or simple caching cannot provide necessary performance or enable data sharing among grid. Databases for instances are predominantly disk bound and centralized and cannot support a distributed grid deployment. Messaging though distributed in nature has no data management abilities and is usually best suited for point-to-point event communications. Simple caching as offered by some grid vendors can provide some relief to grids, but once the scale of deployment increases such approaches fail to meet the data scalability and consistency

requirements. Grid implementations like Globus (www.globus.org) offer mechanisms like GridFTP and Global Access to secondary Storage (GASS), but these do not address the real-time requirements of a grid and cater more to large volume data movement and batch replication. Without a robust data infrastructure in place, fast and reliable data access/distribution is not possible. Under such circumstances, most compute grids will be relegated to being fast running processes with no data to process - like a Ferrari with no fuel!

THE NEED FOR A DATA FABRIC

Given the power of grid computing, the amount of data that an application manipulates is less important than the timely availability and the location of the data. If a set of data must travel from one end of a grid to the other or worse arrive lazily from the enterprise data repository, for example, the potential exists for a considerable amount of wait time, depending on network traffic and the speed of the data source technology.

- Bart Jacob, ITSO Redbooks Project Leader, IBM.

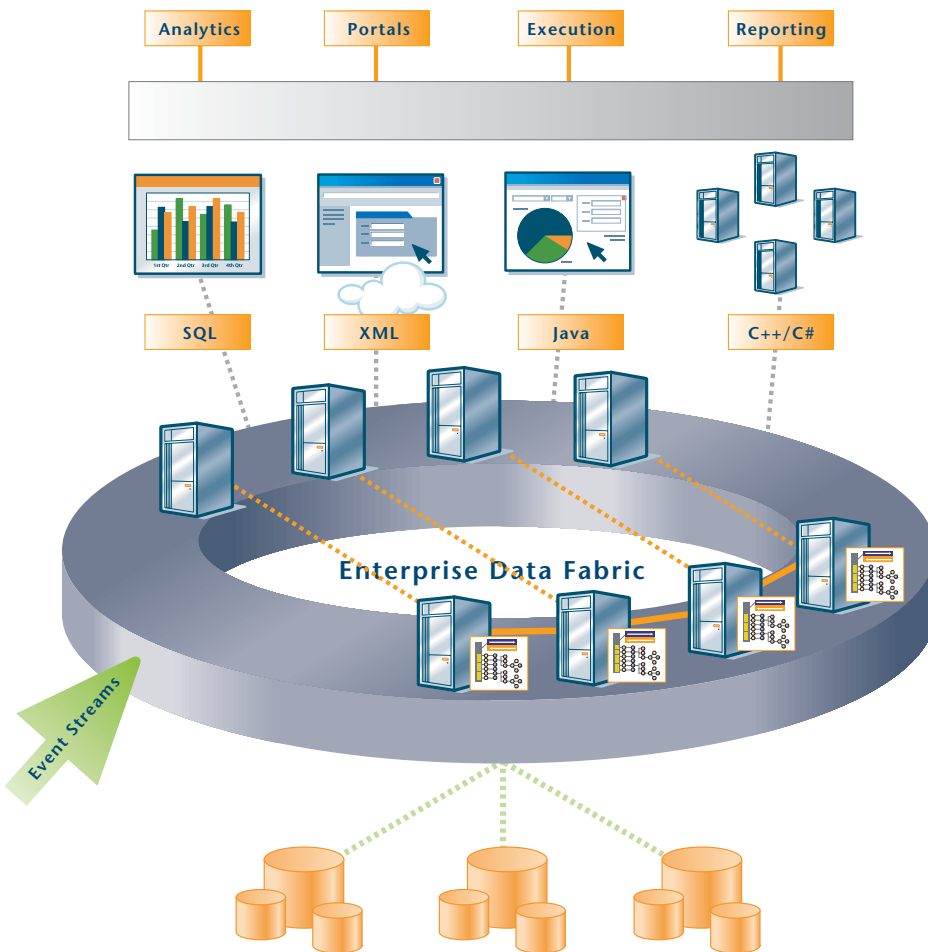
The need for a high performance data fabric in grid deployments is best appreciated if the following aspects about enterprise grid deployments are considered:

- **Computationally intense jobs are inherently data intensive:** Number crunching jobs typically depend on large quantities of data that is managed across many enterprise data repositories that are not grid-enabled. Data access rates and throughput have to be high and guaranteed for the compute job to effectively utilize the available CPUs.
- **Data sources not designed to handle highly parallel load:** Large-scale grid applications employ different strategies to reduce the processing time. One such strategy is to use many independent, concurrent tasks. Here, the number of grid nodes used is highly unpredictable and dynamically decided based on the incoming data. For example, pricing calculations in capital markets use Monte Carlo simulations for randomized scenario analysis and for calculating risk exposure. Greater the number of scenarios, more accurate the calculations. Most of these simulations are executed in parallel and depend on data from multiple sources like security reference data from external sources, portfolio/position data and compliance information. What is often need is a fast, distributed data store that can aggregate all this information and provide instantaneous access to data on any node that runs a simulation. Direct access to legacy data repositories to access the data in real-time is not possible as the legacy sources are not designed to handle such sporadic connection load or spikes in data access.
- **Grid Jobs and Tasks are inter-related:** Many compute intensive jobs use 'parallel batch' strategy where each user's batch work is subdivided into many sub-tasks, concurrently executed on many grid nodes and results collected and aggregated. The aggregated result is

then used by the next job. So, the complete unit of work is like a workflow or a jobflow to be precise. Jobs share data and produce data that trigger the next job(s). Sometimes multiple jobs modify the same data concurrently. The number of nodes and the actual grid nodes on which these tasks run is dynamically decided. This poses many challenges on the grid implementation, which now has to ensure extremely fast replication and routing of data across many grid nodes, intelligently place the right data on the right node and guarantee consistency and integrity of data. To fully utilize a compute power on a node, all the data required for a task should be available in memory on the node. Otherwise, the compute problem would soon turn into an IO bottleneck or a data latency problem.

- **Data managed in the Grid has to be Highly Available:** Whether it is shared static data or data produced as a result of a compute job, it has to be ensured that data is always available and is not lost upon node failures. Data that is managed in memory has to be replicated to multiple member nodes and recovery of data should be quick.

The aforementioned requirements highlight a critical need for a data infrastructure that addresses the shortcomings of current architectures. In the following sections, we will explore how the GemFire Enterprise Data Fabric (EDF) can offer such an infrastructure to support grid-computing environments.



GEMFIRE EDF - AN OPERATIONAL DATA FABRIC FOR GRID COMPUTING

GemFire EDF is a scalable, distributed platform to manage increasing volumes of enterprise data and streaming events with almost zero-latency. With advanced data virtualization, distributed caching and data distribution capabilities, GemFire EDF enables the delivery of actionable information to the right application at the right time.

Figure 1: GemFire Enterprise Data Fabric (EDF)

The GemFire EDF (shown in Figure 1) can be envisioned as a data grid that provides high performance, in-memory data management for a grid environment. GemFire combines memory from physically distributed grid nodes into a single, extensible enterprise-wide distributed cache. This enables any process to reliably share, store, replicate, transform, route, and synchronize large volumes of data across the grid in real-time. GemFire creates the illusion of data locality for grid applications by delivering the necessary data as quickly as possible, even though that data may have been moved from another compute node on the grid or accessed from a data source. Further, grid applications need worry about the location of the actual data source or the format in which data is stored. GemFire delivers the data to the application in a format that the application understands and removes the need to deal with multiple data access and network protocols to connect to different sources. This concept is referred to as location transparency in the world of grid computing.

By reliably caching, synchronizing and integrating dynamic events with static data from multiple data, GemFire dramatically speeds-up query performance, improves resiliency, and conserves system resources. For transactional processing environments, GemFire provides a distributed transaction service that can integrate into J2EE containers, such as WebLogic and WebSphere, through standard interfaces like JTA. This feature helps orchestrate data manipulation operations across grid nodes with no compromise on reliability and consistency.

Grid Data Management Requirements	GemFire Solution
1. Ultra-high performance, low-latency data distribution and high data throughput.	<ul style="list-style-type: none"> • In-memory data management with no disk latency. • Highly parallel data distribution and optimized transportation layer.
2. Caching and persistence models for extremely large volumes of data across distributed environments.	<ul style="list-style-type: none"> • Multiple, flexible data caching topologies. • Distributed data partitioning schemes across servers for dynamic data • Scalability under increasing loads. • Support for distributed transactions to facilitate a grid approach in OLTP environments. • Optional overflow to disk to accommodate excess data on a single node.
3. Data virtualization and location transparency	<ul style="list-style-type: none"> • Pool memory and disk across network to manage large quantities of data and transparently move data on demand. • Access and Synchronize data to/from data sources. • Standards-based querying across multiple data elements.
4. Guaranteed QoS for data access and high data availability for reliable operations	<ul style="list-style-type: none"> • Advanced data placement strategies to collocate data where it is most • Frequently requested and avoid expensive transformations and network hops. • Main-memory replication, or disk persistence to guarantee availability of data.
5. Multiple data format and language support with true interoperability	<ul style="list-style-type: none"> • Native support for multiple data formats - Java, C/C++.
6. Seamless fit into existing enterprise architectures	<ul style="list-style-type: none"> • Standards-based interfaces for easy insertion into distributed environments. • Pre-configured interceptors/drivers for zero code integration with databases.

GEMFIRE EDF: KEY TECHNICAL CONCEPTS

Distributed System Membership: As show in Figure 2, the GemFire EDF consists of any number of member data nodes that are connected to one another in a peer-to-peer fashion, such that each member is aware of the availability of every other member at any time. GemFire's distributed cache API presents the entire distributed system as if it were just one logical cache completely abstracting the actual location of the data or the data source from the developer.

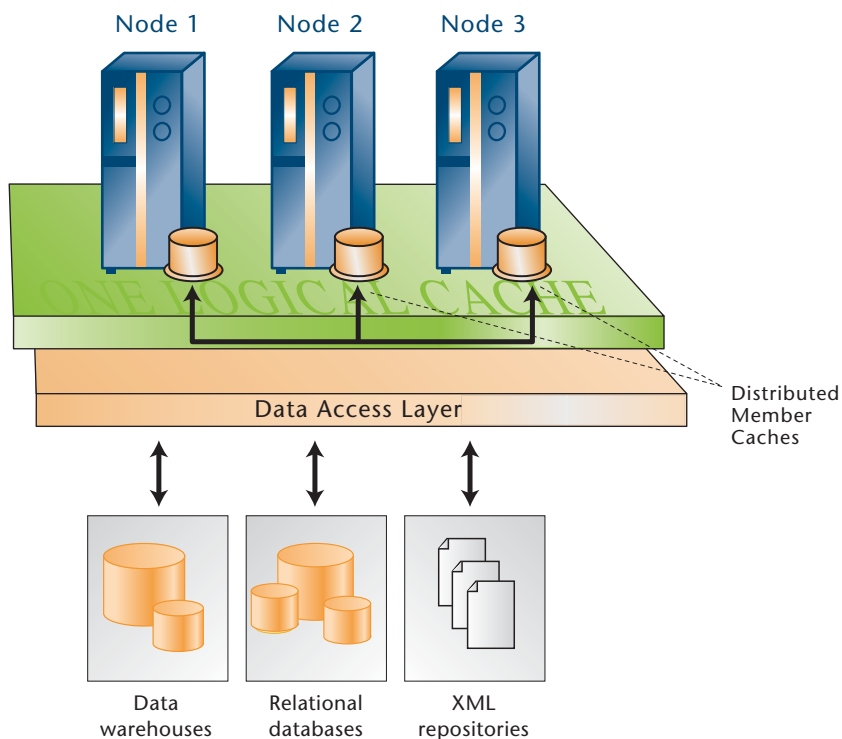


Figure 2: Single logical cache model across multiple grid nodes

GemFire's group membership service uses a dynamic membership discovery protocol to allow members to added or removed dynamically. Each member directly connects and communicates with every other member using point-2-point connections (TCP) or using broadcast communications (Multicast).

Data Regions: Grid data is managed in a logical namespace of data 'regions'. A region is a map of key-value pairs, where the key and value are defined by an application. A region carries many configurable properties that control aspects such as data distribution, data consistency, synchronization with back-end data sources, and disk overflow. Regions typically manage data in main memory, and can be replicated across any number of member nodes for guaranteed availability of data or to allow parallel access to data without unnecessary transfers over the network. Regions can also be partitioned across many members of the distributed system and access to any data object is at most one network hop, ever. The concept of partitioning is further explained in a following section titled 'Dynamic Data Scalability'.

Cache Topologies: To support grid deployments of varying functions and scale, GemFire EDF offers a variety of caching topologies. A combination of different transport protocols and a very efficient distribution layer within GemFire minimizes data copies, context switching and layering and ensures the most efficient movement of data across all these different topologies.

Peer to Peer: In this topology all nodes in the network know about each other. When new nodes join in or existing nodes leave the distributed systems, all nodes in the topology get notified. This allows the users to dynamically adjust the size of the distributed systems.

Client Server: To ensure significant scalability in a grid environment with thousands of client application nodes, the distributed caching system can be organized in a hierarchical topology: a set of member caches that are connected in a peer-2-peer network to form a farm of cache servers and any number of client applications that load-balance to this farm of cache servers. Configuration policies dictate how data propagates between client applications and between cache servers. Each client application can also manage a local cache of the most frequently used data. Policies can be configured to determine how data consistency is maintained across the Grid. Figure 3 illustrates the peer-to-peer and client-server caching models.

WAN topology: With this topology, distributed systems at different geographical sites are loosely coupled through "gateway" processes. Each distributed system has at least one VM that acts as a gateway to distribute data to and from the other sites. Distribution of cached data between sites is transparent to applications. If a remote distributed system becomes unavailable, the other distributed systems continue to operate, and data is stored and forwarded to the remote distributed system when it become available once again.

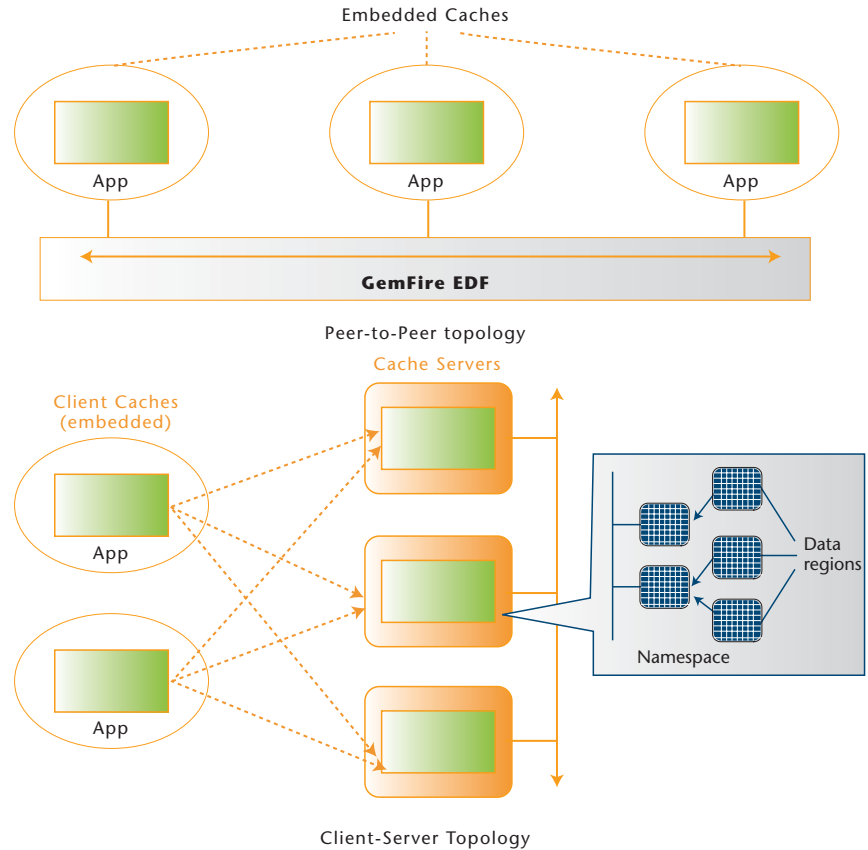


Figure 3: Peer-2-Peer and Client Server Topologies

Distributed Event Notifications: GemFire EDF provides a distributed event notification service where updates to a data region are propagated through a 'listener' framework to any member subscribing to that data region. Unlike an enterprise messaging system, the programming model is very intuitive. Applications simply operate on the object model in the cache without having to worry about message format, message headers, payload, etc. Sharing of data objects combined with event notifications enables tightly coupled parallel programs to be deployed on grids that require a high degree of sharing and synchronization.

Data Consistency: GemFire provides multiple models to ensure data consistency across the grid. Applications can use asynchronous data distribution when strict consistency is not required or when concurrent processes have no data conflicts. Applications that need delivery guarantee, or in other words want to ensure that data has been successfully propagated to other nodes use synchronous distribution with acknowledgements. For applications or scenarios where there is potential for data conflict, a global distributed lock service can be used to ensure a pessimistic data distribution model. GemFire also supports changes to one or more data regions as single atomic unit of work using a built-in distributed transaction service. It provides a simple set of plug-in interfaces for application developers to enable connectivity with remote data sources such as databases, custom applications, etc. A 'Data Loader' callback interface

enables loading data from multiple disparate sources whereas a 'Cache Writer' callback interface enables synchronizing the data changes back to the data sources.

Dynamic Data Scalability: To manage data volumes (in memory) much larger than the process space limit or the capacity of a single machine, GemFire supports a model for automatically partitioning the data into many buckets, managed across many processes, and spread across many nodes. GemFire provides this functionality through a highly available, concurrent and scalable distributed data structure. Applications operate on a HashMap like structure, and under the covers GemFire manages the data across the members of the distributed system, guaranteeing that data access is at most a single network hop. To reduce the need for network hops altogether, data entries can also be stored locally in a disk-based LRU cache (as discussed earlier). New hardware nodes can be dynamically added or removed to alter memory capacity and account for data volume changes without impacting any deployed applications. Data volumes could change over a period of time. For instance, as tasks become more and more complex, the data set size used will begin to grow. If the data growth occurs dynamically, GemFire automatically re-balances data buckets across the grid as new member nodes are added (see figure 4). SQL-like queries submitted to partitioned data regions are routed to relevant member nodes that have a partition, executed in parallel and the union of the results returned to the client. This parallel query execution maximizes CPU utilization on the grid and reduces data access latency. Loading/synchronizing of data from/with external data repositories can also be made parallel from each node that hosts a partition.

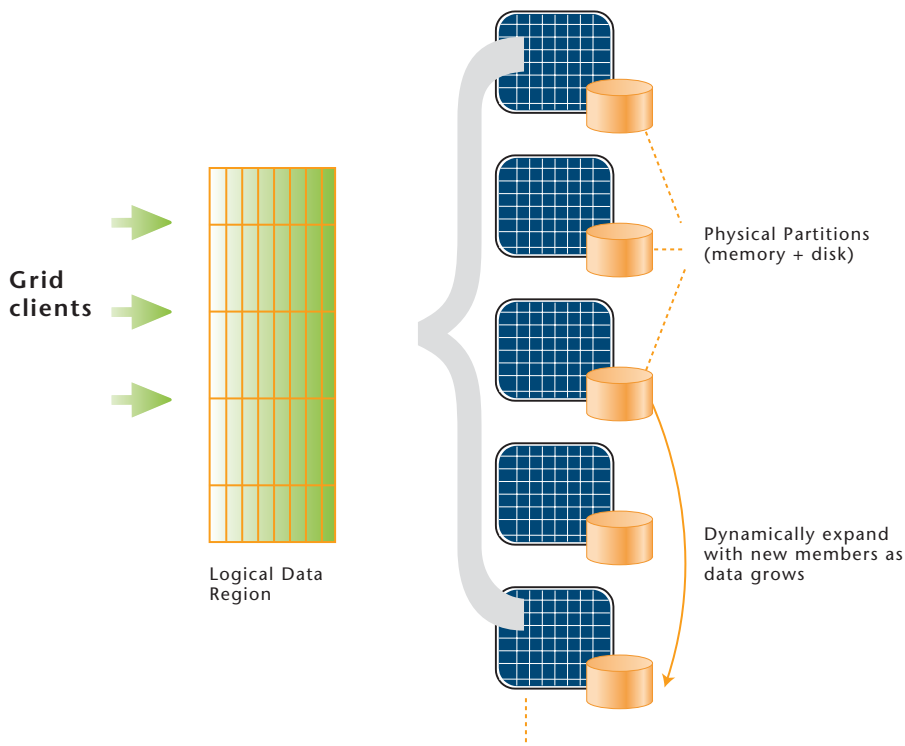


Figure 4: Distributed Data Partitioned across a Grid

High Availability Models: For critical grid operations, it is essential to ensure data availability at all times and avoid single points of failure (SPOF). GemFire provides highly availability through replication to one or more "mirror" (backup) cache nodes. A mirror synchronously receives all events on a cache across the entire distributed system guaranteeing 100% backup of data at all times. It acts as a standby, and is used to load data into the primary cache when a failed application restarts. The most common deployment model is one where the cache is co-located with the application in the same process. If the application process fails for any reason, the cache gets automatically disconnected from the GemFire system. Upon application restart the cache reloads itself from the backup (lazily or at startup). Making data high availability through in-memory replication though efficient may not suffice in some situations. Certain applications managing critical information in the cache may mandate that data be reliably managed on disk. GemFire accommodates such applications by optionally persisting data held in the cache to disk (file system). This way complete data recovery is guaranteed even under scenarios involving complete application(s) failure.

GEMFIRE EDF SYNERGY WITH OTHER GRID TECHNOLOGIES

The ability of an EDF to position and distribute data in a dynamic fashion opens doors for intelligent interactions with other technologies in a grid ecosystem as discussed in the following.

GemFire and Compute Grid Engines

GemFire EDF complements compute grid engines like Platform Symphony and DataSynapse GridServer by serving as a high-speed operational fabric for storing, transporting, synchronizing, and reusing data specifically routed to individual compute task nodes. Without such a high performance data fabric, grid applications degenerate into problem set domains which require little or no data, or problem sets which require manual ftp-like solutions to the transportation and synchronization of compute data. GemFire creates transparent access to shared data that can be easily load-balanced across a grid, providing distributed caching to support both intra-node and inter-node collaboration.

GemFire EDF can provide the following benefits to compute grids:

- **Data-Aware Routing.** A grid scheduler connected to GemFire can manage the initial distribution of data among processing units to ensure that required data is present or en-route as jobs are dispatched. As tasks finish, the scheduler can either direct new jobs to processing units that are already equipped with the right data, or it can initiate additional data distribution as needed. A scheduler able to exploit detailed knowledge of data locality can minimize both the total data distribution load on the network by sending only needed information instead of supersets, and it can maximize processor utilization by reducing the time processors must spend waiting for data.

- Computation Control.** Grid processing nodes plugged into the GemFire EDF can share intermediate results and progress metrics with one another and with the job scheduler. This enables fine-grained global adjustment of computational tasks for maximum efficiency. In the context of Monte Carlo simulations, early and efficient feedback may make it possible to eliminate sub-optimal execution and save time. For example, the job scheduler might elect to forego assigning certain tasks, or it could even choose to abort unpromising jobs already in progress. Furthermore, application code can be modified to distribute progress information via GemFire on a periodic basis. This information sharing enables the job scheduler to build detailed knowledge about specific server characteristics during early processing runs, so that subsequent task assignments can be made with increasing intelligence.
- Non-blocking Interactions with Clients.** It is undesirable for clients to block while waiting for long Grid jobs to finish. This situation can be easily avoided when a grid client is connected to the GemFire EDF. The client need only spawn a separate thread that waits on a GemFire cache listener, which will be triggered by the scheduler on job completion.

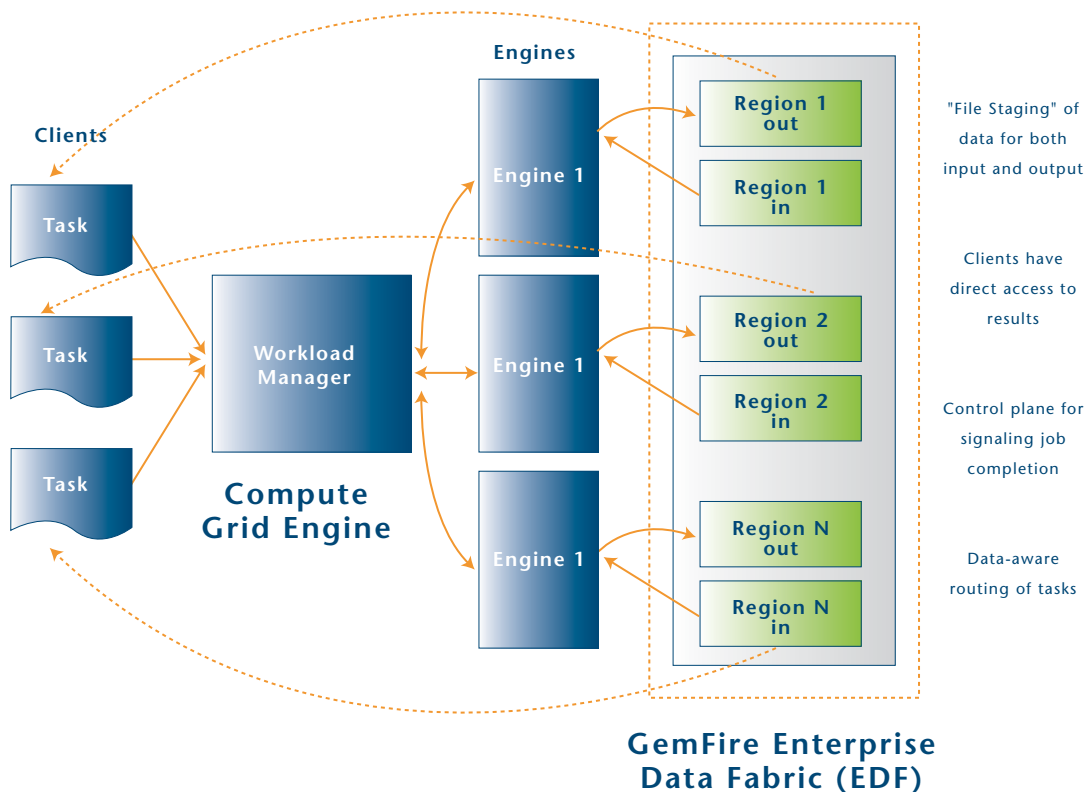


Figure 5: GemFire Data Grid and Compute Grid Synergy

GemFire EDF and Grid Provisioning Software

With complete knowledge of the grid hardware and application deployment, a grid provisioning engine such as SUN N1 provisioning engine, VMWare or the IBM Tivoli (ThinkDynamics) provisioning software can also be used to provision the right number of GemFire cache servers with a priori knowledge of job/application schedule.

Data transfer can be scheduled into the grid so the data arrives on the appropriate location at the time when it is needed. For instance, if a data transfer takes one hour and the data is required for a task slated to start at 2:00 a.m., then data transfer can be scheduled in advance so that it is available by the time the job requires it. Also, factors such as load concurrency, amount of data required by jobs and 'Quality of Service' policies will dictate the hardware requirements such as CPU power, memory and disk requirements. With this advance knowledge, virtualization and provisioning software can bootstrap GemFire on a set of grid nodes before the grid jobs are launched. 'Just-in-time' provisioning of data for data intensive grid tasks optimizes the resource usage without comprising on data access performance.

GRID JOB FLOW EXAMPLE WITH GEMFIRE

Consider a computational job flow, where a job is a unit of work and the job flow encompasses several such jobs that execute in some serial order to complete a specific function. Each job operates on a subset of the data and requires several parallel activities or tasks to complete. The computational tasks will often require static reference data from one or more enterprise databases plus some real-time data (called 'job data'). This job data is supplied by the initiating client application and provided just-in-time before the job commences. The compute jobs/tasks are provisioned onto a grid environment and data is also provisioned into the same environment using GemFire.

A typical deployment of a grid data cache using GemFire will be on a farm of grid nodes, each running a GemFire cache server (discussed under client-server caching in the GemFire EDF technical concepts section). Depending on the amount of data in use, the concurrency and access rate requirements, data will either be replicated across the farm, partitioned or both. The cache servers will be provisioned by a grid provisioning service and the nodes to host the data services will be selected based on the memory and disk requirements for managing the total data. The cache will be bootstrapped with the static data elements prior to start of the job flow.

The following steps describe the data architecture and how the grid scheduler, the application and GemFire EDF interact to complete the job flow.

- 1) The job data set is typically made available just-in-time. The client application directly publishes the data into GemFire EDF from any remote node. This data is routed to one cache server and replicated to others as configured. In some cases, where the data set size is very large to fit in process memory, the data is partitioned across the cache server farm.

- 2) The client application uses the grid scheduling engine to schedule the first job. The scheduler's resource management layer divides the job into a set of parallel tasks and initiates them on several grid nodes.
- 3) As part of the initialization, the worker (task) process connects to the cache server farm and initializes its local cache with some subset of data.
- 4) Each compute connects to the cache server farm process in a similar fashion. Requests to the cache servers are automatically load-balanced.
- 5) The compute tasks fetch data from one or more of the cache servers as needed. If the data is partitioned across cache servers, GemFire automatically locates the right partition and returns the data to the task.
- 6) Multiple compute tasks can synchronize with each other using distributed locks and using event notifications. The tasks publish their results in the cache, where they get aggregated.
- 7) Once the job is completed, the client application fetches any required results directly from the cache rather than having the task serialize data to the client application through the scheduler engine.
- 8) The next job initiated operates on a new data set and the results obtained from the first job. All the required data is readily available for instant processing.

CUSTOMER CASE-STUDY - RISK ANALYTICS DATA GRID

The Problem: The practical application of an EDF is better appreciated by the following use-case, which pertains to the risk calculation grid of a leading investment bank. Prior to the adoption of GemFire EDF, this firm was facing significant latencies (more than 8-hours) in its overnight risk calculations and often faced the predicament of not being able to complete these prior to start of trading the subsequent morning. This put them at a risk of substantial monetary loss. The long cycle time was primarily due to a surge in data volumes that were required for these risk calculations. Volatile market conditions as well as compliance requirements, both of which required additional sources of information to be included in the risk calculations, were the major reasons for this increase in volume. An architectural analysis revealed that data latency and data distribution accounted for about 70-80% of the risk computation cycle time.

The GemFire Solution: GemFire EDF was deployed as a data grid solution to bolster this risk computation infrastructure. As can be seen in Figure 6, the GemFire EDF deployment supports two compute infrastructures: a) the pre-processing and theoretical value computation grid and b) the risk calculations. These two grids account for roughly 2 billion calculations across

multiple securities and portfolios. From a data flow standpoint, market data snapshots and securities data are pushed into the data grid and held in data regions that are replicated for high availability. These first-level data regions handle large volumes of fast moving data and serve-up relevant subsets of data to a set of embedded caches (smaller data volumes) that are collocated with the pre-processing compute grid. Such a hierarchical network of caches also supports instantaneous sharing of common data elements as well as intermediate results across multiple applications that repeatedly enrich the data. The results from the pre-process grid are moved to the risk calculation data grid that has a similar structure with a set of replicated large data volume cache servers supporting embedded caches that provide local data access to risk applications. GemFire also enables disk persistence of all information held in the in-memory data fabric for complete data recovery.

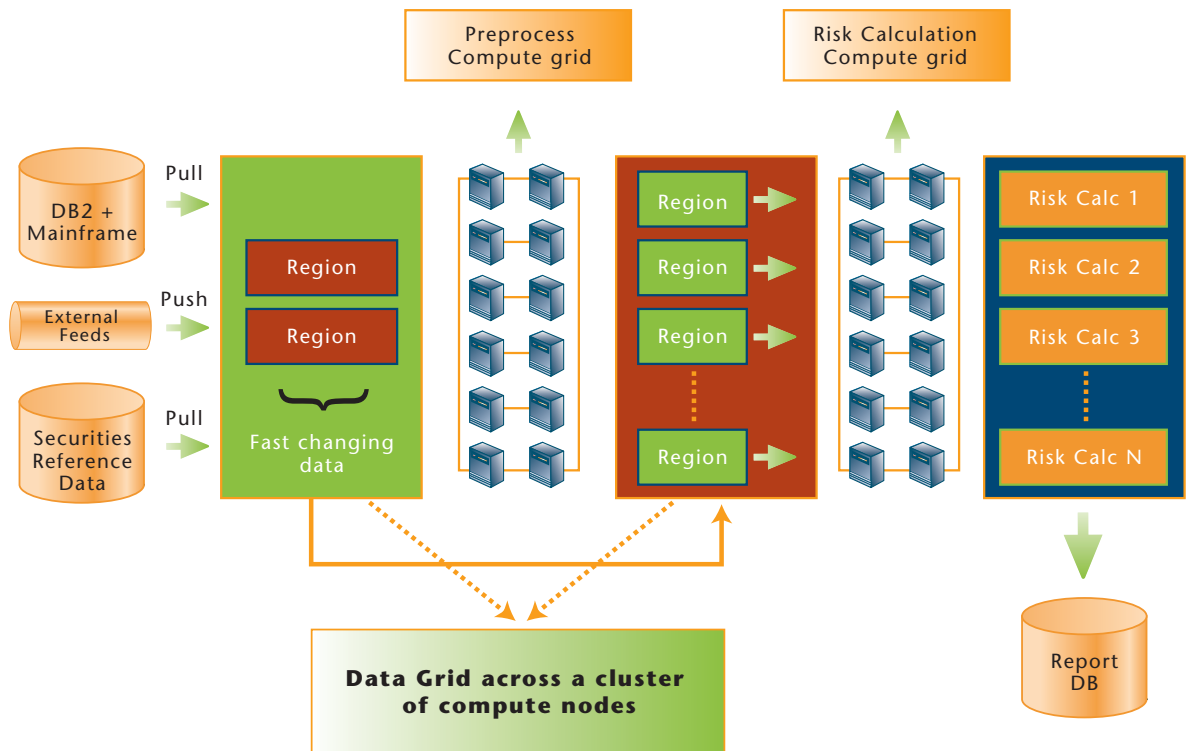


Figure 6: GemFire EDF in a Risk Computation Infrastructure

Through intelligent data caching, distribution and replication mechanisms, GemFire EDF increases data availability for the risk calculation applications and provides them instant access to relevant information. By positioning the data close to the consumers (risk calculators) through a distributed network of data regions, the data latency issues are successfully resolved.

This infrastructure is deployed on a distributed Linux Architecture using RDMA based Infiniband topology. GemFire ideally complements the RDMA by distributing data across interconnected nodes (e.g. blades), creating a single system image of shared data that elegantly spans the system area network created by Infiniband's hardware virtualization technology. By virtue of sharing state in a distributed manner across tightly interconnected blades, GemFire provides massive scale out, business continuity, and load balancing of data resources.

Benefits:

- Reduction in risk computation cycle from 8+ hours to less than 2 hours - ability to run risk computations more often (intra-day).
- Access rates of around 350,000 reads/second, which translates to improved compute speed.
- Increased accuracy through the ability to handle larger data volumes, greater number of iterations and more diverse data entities that facilitate richer risk calculations.
- Greater scalability (more compute nodes) and high data availability through a replicated, in-memory risk data layer

SUMMARY

As grid computing continues to grow as a popular paradigm in most IT organizations, there is an obvious need to expand most pilot grid implementations to a larger scale to support several enterprise business operations. But, such expansion plans are often stymied by data latency bottlenecks (as outlined in the recent report by the 451 group) that cripple grid processes. Given these challenges, grid deployments need a reliable information infrastructure - an enterprise data fabric that is:

- pervasive and distributed,
- able to virtualize information and provide location transparency,
- characterized by extremely low data latencies,
- capable of very high throughput,
- highly available and durable,
- able to manage transactions across distributed nodes.

A data fabric implementation is no simple task either. As outlined in this white paper, the location of where data is being managed, data distribution, data consistency and high availability are important considerations in such an implementation. GemStone Systems has the required expertise to handle these challenges and offers a best-in-class grid data management solution powered by the GemFire EDF - a proven entity in mission-critical environments. Learn more about GemFire EDF at <http://www.gemstone.com/products/gemfire/>.



Corporate Headquarters:

1260 NW Waterhouse Ave., Suite 200 Beaverton, OR 97006 | Phone: 503.533.3000 | Fax: 503.629.8556 | info@gemstone.com | www.gemstone.com

Regional Sales Offices:

New York | 5 Penn Plaza, 23rd Floor, New York NY 10001

Washington D.C. | Phone: 301.325.8405

Santa Clara | 2880 Lakeside Drive Suite 331 Santa Clara, CA 95054 | Phone: 408.496.0242

Copyright© 2008 by GemStone Systems, Inc. All rights reserved. GemStone®, GemFire™, and the GemStone logo are trademarks or registered trademarks of GemStone Systems, Inc. Information in this document is subject to change without notice.