Distributed Data Management with VMware vFabric GemFire

Real-Time Data Correlation: Latency and Sustained Operations

TECHNICAL WHITE PAPER
# Table of Contents

Abstract .................................................................................................................. 3  
GemFire: A Quick Look at Terms and Topology ....................................................... 4  
Sources of Data Latency and How GemFire Helps Reduce Them ....................... 7  
Data Consistency ..................................................................................................... 7  
Transactions and Sharding ...................................................................................... 8  
Data Subscriptions .................................................................................................. 9  
Sustained Operations in the Face of Failures (Machine, Network, Memory) ............ 9  
Machine Failure Data Protection ........................................................................... 9  
Site Failure ........................................................................................................... 9  
Memory Issues ....................................................................................................... 10  
  Resource Manager .............................................................................................. 10  
Handling Network Splits and Outages with Split-Brain Detection ......................... 11  
  Surviving Side .................................................................................................... 13  
  Losing Side ....................................................................................................... 13  
  Pipeline Processing ............................................................................................ 13  
  GemFire Membership Role .............................................................................. 13  
Slow Receivers (Order, Latency) ........................................................................... 15  
Administrative Functionality .................................................................................. 15  
  GemFire Administrative API ............................................................................ 17  
  Retrospective Analysis ..................................................................................... 18  
Network Issues ...................................................................................................... 18  
  Socket Control ................................................................................................ 18  
  IPV 4 vs. IPv6 ................................................................................................. 18  
Disk Issues ............................................................................................................ 18  
  Auto-Compaction of Operations Log ............................................................... 18  
  Handling Catastrophic Loss of Storage Disk Data ........................................... 19  
  Online Backup ................................................................................................. 19  
Conclusion ............................................................................................................ 19  
Find Out More ....................................................................................................... 19
Abstract

VMware vFabric™ GemFire® is in-memory distributed data management platform that pools memory (along with CPU, network and optionally local disk) across multiple processes to manage application objects and behavior. Using dynamic replication and data partitioning techniques, GemFire offers continuous availability, high performance and linear scalability for data intensive applications without compromising on data consistency, even under failure conditions. In addition to being a distributed data container, it is an active data management system that uses an optimized low-latency distribution layer for reliable asynchronous event notifications and guaranteed message delivery.

The goal of this white paper is to look at two key issues for a 7x24x365 high-speed ingest and analysis system. The first issue is maximizing performance by reducing overall system latency. The second issue is maintaining system operations despite inevitable failures and/or shortages of RAM, network connectivity and whole systems.
GemFire: A Quick Look at Terms and Topology

To put the rest of this paper in context, we will first review some key GemFire concepts. From a topology perspective, GemFire supports several deployment styles. A peer-to-peer topology is the building block for the others. In a peer-to-peer GemFire system, each member holds one or more sockets open to all the other members. This topology supports running application code and data in some or all of the members. As new members join the distributed system, they discover the others through the locator process.

Figure 1. Peer-to-peer topology
GemFire also supports a client-server topology where clients connect to servers in the distributed system. In this topology, the data resides in the servers, with an optional L1 cache in the clients. Application code may run in clients, servers, or both.

Figure 2. Client/Server topology
GemFire also supports a multi-site topology. In a multi-site configuration (generally run across a wide area network), distributed systems are configured to communicate with one another through specially configured gateway members. Each system is its own distinct distributed system and often each system also acts as a server system in a client/server configuration. The members within each distributed system operate among themselves in standard peer-to-peer fashion. Additionally, the gateway members distribute cache operations to the remote distributed system sites and receive distributions from them.

The other key GemFire concept discussed extensively throughout this paper is that of Regions. A GemFire data region is a logical grouping within a cache for a single data set. You can define any number of regions within your cache. Each region has its own configurable settings governing elements such as the data storage model, local data storage and management, data event distribution, and data persistence. There are four types of regions, and a single member can have any or all of them:

- **Partitioned** - Server system-wide setting for the data set. Data is divided into buckets across the members that define the region. For high availability, configure redundant copies, so that each data bucket is stored in more than one member, with one member holding the primary. GemFire allows additional members to be added to the distributed system and host another part of the partitioned region. This provides the ability to dynamically grow the total size of the partitioned region.

- **Replicated (distributed)** - Holds all data from the distributed region. The data from the distributed region is copied into the member replica region. Replicated regions can be mixed with non-replicated, with some members holding replicas and some holding non-replicas.

- **Distributed (not replicated)** - Data is spread across the members that define the region. Each member holds only the data it has expressed interest in.

- **Local (not distributed)** - The region is visible only to the defining member. Typically used by GemFire clients for communication region to hold server data and/or messages.
Sources of Data Latency and How GemFire Helps Reduce Them

Most data latency in applications comes from electronic delay (disk I/O, network I/O) or data impedance mismatch. Electronic delay is the result of data not being in process. The slower it is to retrieve the data from its resting place, the further the distance, the more processes the data must pass through, and the more transformations the data must undergo, the more latency the requesting application experiences.

GemFire provides a number of mechanisms to avoid these sources of latency. First, GemFire can enable processes to run in the same process memory as the data. For data-driven applications, GemFire provides cache listeners that enable program logic to run as data arrives in process in GemFire. This often works well for applications that are taking in a large amount of data and acting on each piece of data as it arrives. For applications that need to retrieve data and then run some code, replicated regions provide a means to manage small (20-30GB) amounts of data in an application process. For larger data sets, GemFire uses partitioned caching to keep the data no more than one network I/O away.

Another alternative for applications that need a large amount is to push the business logic to the data using the function services in GemFire. This enables an in-memory, distributed, map-reduce operation to act on the data. These options are not mutually exclusive. For example, a piece of data arrives that results in a calculation that needs to compare this data with a large set. This could cause the listener to use the function service to execute a function across a partitioned region. That function might need to look at a reference table in the course of those comparisons. That reference table could be stored in a replicated region that was collocated with each of the partitions of the partitioned region.

When writing data into a region, GemFire’s default behavior is to avoid disk. Disk access is tens of thousands of times slower than memory access; even memory that is one network “hop” away. GemFire protects data by writing it to two or more computers either with replicated regions or with replicated buckets (for partitioned regions). To minimize latency, network bandwidth usage, and to minimize garbage collection, GemFire enables updates to propagate just changes to an object rather than the entire object.

There are many times when it is just not feasible to have code running in the server processes (for example, when many applications share a set of data at the server level). In this case, the best option is to have data no more than one network hop away, in main memory of one of the server processes. GemFire always provides a way to do the most efficient data access possible given various constraints. When clients connect into a distributed system they do so region by region. For partitioned regions with n-1/n pieces of data, where n is the number of partitions, data is two network hops away. GemFire provides a capability that uses more sockets but reduces latency, called data-aware routing. With data-aware routing, a client request is routed to the exact computer that hosts the partition that has that data, minimizing latency. To date, we have not seen another client mechanism from an open source or COTS vendor that supports data-aware routing of clients to the exact partition that hosts the data they are fetching.

To avoid data latency caused by using JNI for C, C++, or .Net applications, GemFire provides native client libraries for those languages.

Data Consistency

A data management platform that dynamically runs across many machines requires a fast, scalable, fault-tolerant distributed system as its foundation. It is well known, however, that distributed systems have unavoidable trade-offs and notoriously complex implementation challenges. For example, the CAP Theorem, introduced at PODC 2000 by Eric Brewer® and formally proven by Seth Gilbert and Nancy Lynch in 2002®, stipulates that it is impossible for a distributed system to be simultaneously: consistent, available, and partition-tolerant. At any given time, only two of these three desirable properties can be achieved. Hence when building distributed systems, design trade-offs must be made.

---

Driven by awareness of CAP limitations, a popular design approach for scaling Web-oriented applications is to anticipate that large-scale systems will inevitably encounter network partitions and to therefore relax consistency in order to allow services to remain highly available even as partitions occur. Rather than fail and interrupt user service upon network outages, many Web user applications (e.g., customer shopping carts, e-mail search, social network queries) can tolerate stale data and adequately merge conflicts, possibly guided by help from end users once partitions are fixed. Several so-called eventually consistent platforms designed for partition-tolerance and availability—largely inspired by Amazon, Google, and Microsoft—are available as products, cloud-based services, or even derivative, open-source community projects.

In addition to prior art solutions from Web 2.0 vendors, the GemFire team’s perspective on distributed system design is further illuminated by 25 years of experience with customers in the world of financial services. In stark contrast to Web-oriented applications, financial applications are highly automated and consistency of data is paramount. Eventually, consistent solutions are not an option. Thus, in practice, financial institutions place a mandate on consistency and availability—they want to have their proverbial cake and eat it too. So given CAP limitations, “How is it possible to prioritize consistency and availability yet also manage service interruptions caused by network outages?”

GemFire adopts a shared nothing scalability architecture where data is partitioned onto nodes connected into a seamless, resilient fabric capable of spanning processes, machines, and geographic boundaries. By connecting more machine nodes, GemFire scales data storage horizontally. Within a data partition data entries are key/value pairs, with thread based, read-your-writes consistency.

The isolation of data into partitions creates a service-oriented pattern where related partitions can be grouped into abstractions called service entities. A service entity is deployed on a single machine where it owns and manages a discrete collection of data—a holistic chunk of the overall business schema—hence; multiple data entries collocated within a service entity can be queried and updated in a transactional manner, independent of data within another service entity.

**Transactions and Sharding**

The ability to partition or shard the data is not a latency issue but a scaling and data consistency issue. GemFire provides a number of mechanisms to provide for data consistency in a partitioned or replicated system. GemFire partitioning provides for dynamic growth by adding new machines to host elements of the partition. GemFire also allows the region to be dynamically rebalanced. To provide maximum flexibility, GemFire has its own hash based sharding algorithm that supports range based partitioning or allows developers to insert their own algorithm so they can more tightly control which data is on which computer. Many other open source or COTS partitioning systems either do not allow for dynamic system growth or fail to provide a means for developers to control data placement.

To provide for high availability for a data element in a partition, GemFire allows for a designer to specify a number of replicate copies of the data that will be provisioned on different computers. To provide for read/write consistency of those copies, one of them is designated the primary. Unlike most other systems that allow for partitioned read/write consistency, GemFire allows client applications to see and use the secondary copies of that data. Data reads are directed to any one of the copies, thus helping to insure scalability and performance. Writes, on the other hand are directed through the primary copy only. A write on the primary copy results in a blocking update to all of the secondary copies.

GemFire has a built-in transaction coordinator. GemFire supports transactions on data entities within a partition to allow for updates to multiple related objects to be handled in an ACID manner. Transactions can span partitioned, collocated regions and replicated regions. Many other open source and COTS products do not support transactions or rely on the slow and cumbersome two-phase commit approach.

---

Data Subscriptions
To provide a client application a consistent view of the distributed system GemFire provides several choices for clients to subscribe to data in a distributed system. In many distributed caching or in-memory DBMS style systems (Mongo, TerraCotta, etc.) if data changes on the server the client is unaware of those changes. Some systems let you know if data you currently have has changed. Only a few systems (all COTS) let you get an initial image and register interest by pattern or query in a “potential” data set of interest. And only GemFire lets you do all of this in a fault tolerant manner.

GemFire clients provide for notification of all changes to data retrieved from the server side. Additionally, clients can get an initial result set and register interest in data by keys, by regular expressions on keys (where those key must be strings), or create a Continuous Query using language statements on the data. GemFire uses the Object Data Management Group (ODMG) Object Query Language (OQL) a subset of SQL-92 with some object extensions, to query and register interest in data stored in GemFire. Furthermore, in the event of a computer failure in the distributed systems (the server side), GemFire will automatically reconnect clients to an alternate distributed system member, which has a redundant copy of their registered interest.

Sustained Operations in the Face of Failures (Machine, Network, Memory)
Beyond low latency GemFire has two main goals: consistency and availability. The main enemy of availability is the inevitable failure of computers and networks. For distributed memory solutions, the other challenge can be, “what do I do when I run out of memory?”

Machine Failure Data Protection
For distributed memory based solutions the first line of defense against the failure of a single computer is to have a secondary (tertiary, etc) copy of the data on some other computer. GemFire provides two ways to do that. Regions can be replicated and thus a full copy of the data set is on multiple computers. Or regions can be partitioned with a configurable number of read-only secondary copies.

Site Failure
In the event of total power failure at a site no local data replication scheme will protect from site level loss of all data in memory. GemFire provides two options to help deal with that issue. First, on a per region basis, GemFire provides for optional disk persistence. GemFire has a high performance round robin transaction log that is parallelized for partitioned caches. This log is used in conjunction with background processes that write the data onto the disk partition. With local disk at each computer, we generally see only 5-7 percent performance degradation to throughput for enabling disk persistence. Upon restart, startup occurs with automatic initialization from the most recent data on disk so you do not need to track which members have the most recent data. GemFire also includes all of the start utilities you might expect from a DBMS: command-line management of disk file verification, compaction, backup, and detection of missing/corrupt files.

When continuity of operations is critical, persistence to local disk is just not enough. GemFire has a data replication capability called the WAN Gateway. WAN Gateway enables GemFire, on a per region basis (like most features in GemFire); to replicate data from one distributed system to another. This capability supports multiple sites concurrently. It is a highly available architecture with no single point of failure. Data is queued at the Gateway and sent in optimized batches to all of the other Gateways that are interested in that data region. The queues are memory based with optional overflow and/or persistence to disk. In the event of a network failure, the data queues up until the network recovers.

It is possible to specify multiple network paths for the data. It is also possible to enable conflation in the queues. Doing so results in only the last change to any object being propagated over the WAN. For instances where systems might be network bandwidth limited or the network is down for a long enough period that the amount of data in the queues will take a long time to send, conflation can seriously reduce the total amount of data sent. The WAN Gateway also supports GemFire delta propagation as discussed in the section on latency. The WAN Gateway can be deployed in both active/passive architectures where each site owns its subset of data and active/active architectures.
Memory Issues
GemFire provides several mechanisms to help manage the amount of memory being used for data management and to alert customers when a system will need more memory to maintain a rising amount of data. First, it is possible to specify the total amount of memory a region will use on a system member. This can be done as an absolute number or as a percentage of the available Java heap. Second, as memory reaches this predefined point it is (optionally) possible to specify an eviction action, which will either remove the least recently used object from memory or move it to a disk overflow area. Items in disk overflow do take up some memory as the key and a disk pointer are still left in memory. Third, for partitioned regions, it is possible to scale the system to add more memory at runtime by adding more system members. If actively monitoring memory usage though GemFire’s administration API’s then IT staff will have plenty of advanced indication of the need to add more GemFire instances to accommodate a rising data load.

Resource Manager
To further help with memory management issues GemFire includes a unique Resource Manager to help control memory use in a configurable manner and to alert administrators as memory reaches certain thresholds. The GemFire resource manager utilizes a garbage collection (GC) mechanism to control heap use and protect your virtual machine from hangs and crashes due to memory overload. The manager prevents the cache from consuming too much memory by evicting old data and, if the collector is unable to keep up, by refusing additions to the cache until the collector has freed an adequate amount of memory.

The resource manager has two threshold settings, each expressed as a percentage of the total tenured heap, and both of which are disabled by default.

• **Eviction threshold** – Above this, the manager orders evictions for all regions with eviction-attributes set to lru-heap-percentage. This prompts dedicated evictions, independent of any application threads and it tells all application threads adding data to the regions to evict at least as much data as they add. The virtual machine garbage collector removes the evicted data, reducing heap use. The evictions continue until the manager determines that heap use is again below the eviction threshold.

• **Critical threshold** – Above this, all activity that might add data to the cache is refused. This threshold is set above the eviction threshold and is intended to allow the eviction and GC work to catch up. This virtual machine, all other virtual machines in the distributed system, and all clients to the system receive LowMemoryException for operations that would add to this critical member’s heap consumption. Activities that fetch or reduce data are allowed.

![Figure 4 Tenured memory thresholds](image-url)
Handling Network Splits and Outages with Split-Brain Detection

The issue of network splits is not handled by most distributed data systems. This section explains how the unique (and optional) split-brain detection capabilities of GemFire can keep distributed systems from splitting into two separate running systems when members lose the ability to see each other, as shown in Figure 5. The typical cause of this problem is a network failure. During a network failure, or when partitioning occurs, the problem could result in data inconsistencies or a forced disconnect.

The solution for this problem is to stop one of the two subgroups from continuing to operate independently. Handling network outages is based on the participation of a lead member and a group management coordinator (the locator). The coordinator is a member that manages the entry and exit of other members of the distributed system. With network partition detection, the coordinator is always a GemFire locator. The lead member is always the oldest member of the distributed system that does not have a locator running in the same virtual machine and is not using the administrator interface. The two situations that cause GemFire to declare a network partitioning condition are:

• If both a group coordinator and the lead member abnormally leave the distributed system within a configurable period of time, the caches of members who are unable to see the locator and the lead member are immediately closed and disconnected. Only abnormal loss of the locator and lead member causes GemFire to declare a network partition. If a lead member’s distributed system is disconnected normally, GemFire automatically elects a new one and continues to operate. If a locator is disconnected, a secondary locator takes over.

• If no locator can be contacted by a member, the member closes its cache and disconnects from the distributed system. Since only locators can make membership decisions, a member that cannot contact any locator cannot know if it is isolated from the lead member.

Network partitioning handling allows only one subgroup to form and survive. The distributed systems are disconnected and the caches of other subgroups are closed. When a shutdown occurs, alerts are generated through the GemFire logging system, explaining to administrators what action, if any, to take.

For the algorithm based on the rule stated above to work, the following is critically important to control system startup such that these two roles:

• Do NOT reside on different sides of a crucial shared network component (such as a switch) that could leave each of them on one side of the partition along with some number of other GemFire peers. This would give any IT Operations team a SPLITTING headache!

• Do NOT reside on the same host, in which case a NIC card failure would shut down the entire cluster.

The way to make the algorithm work is to place half of your capacity on one side of whatever components you’re concerned about (those whose failure would cause a network partition, like a major switch, or a MAN link between primary and secondary data centers), and the other half on the other. The placement of the two “special” members makes the surviving side a deterministic outcome.
1. Locators see the lead member leave and remain active.

2. Members lose sight of all locators and are closed because there is no one to make membership decisions. Each member generates an alert. They cannot tell whether the lead member is alive and do not know who the current lead member may be.
1. Failure detection uses member-timeout to remove lost members.

2. On the losing side, each member sees the loss of the coordinator and lead member and shuts down, causing each member and the locator to generate an alert.

3. On the surviving side, members still see the lead member and locator, and remain active.

**Surviving Side**
GemFire provides a complete system to notify system administrators or others of the condition of the system. The group coordinator (locator) virtual machine on the surviving side can have an `AlertListener` configured. Members on the surviving side can have `SystemMembershipListener` configured to processes `memberCrashedEvents` for the peer virtual machines on the losing side.

**Losing Side**
A member that detects a network partition and does not determine it is on the surviving side, disconnects its distributed systems and closes its cache. If a network partition caused the loss, the processes in the other partition (eligible coordinator, lead member, and processes still able to see them) continue to run, electing a new coordinator if necessary. Any clients to the system are unaffected, although they may reconnect automatically to a member on the surviving side.

**Pipeline Processing**
An alternative to split brain detection for applications that have subsets of computers that can run as independent groups is to use the Membership role capability in GemFire. This capability also handles issues of system relationships between groups of computers in areas such as pipeline data processing.

**GemFire Membership Role**
GemFire membership roles give you the ability to decide under what circumstances a distributed system can continue reliable operation after a disruption such as a network failure. A membership role describes how a system member relates to other members, or what purpose it fills.
Members can establish interdependent relationships among themselves by playing one or more membership roles. Membership roles are optional, so they are not required for every member in the system. You configure membership roles at the member level and the region level. First, identify all the roles your system members play (e.g., Producer, Consumer, or Backup). Then decide what roles a given member fills, and add those roles to the member’s system-level configuration. When the member connects to the distributed system, it declares those roles.

Then, for each region, decide which of those membership roles must be present somewhere in the distributed system to enable reliable access to that region. You configure the region’s role requirements and appropriate actions to take when the distributed system loses or regains one of those required roles. With membership roles, you can configure a cache so that one or more of its regions requires another system member to be present in order for the region to be used reliably. You can set up inter-member dependencies so that processing can be halted or altered when dependencies are not satisfied. A member can play multiple roles, and different members can declare that they play the same role. Typical role names are Producer, Consumer, and Backup.

You can specify membership roles to help you identify processes and what domain functions a process performs. GemFire exposes APIs to allow systems to see what roles they play or administrators to see what systems are running with which roles.

The loss-action table below outlines what actions can be set for when a member with a given role is not available to another system that requires such a role be present.

### loss-action

The **loss-action** setting specifies how access to the region is affected when one or more of the roles specified in the region’s membership attributes are offline and no longer present in the system membership. This table defines the options for loss-action.

<table>
<thead>
<tr>
<th>SPECIFICATION (DEFAULT IN BOLD)</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>full-access</td>
<td>Access to the region is unaffected when required roles are missing.</td>
</tr>
<tr>
<td>limited-access</td>
<td>Only local access to the region is allowed when required roles are missing. All distributed write operations on the region will throw <code>RegionAccessException</code> while any required roles are absent. Reads which result in a <code>netSearch</code> behave normally, while any attempt to <code>load</code> is not allowed.</td>
</tr>
<tr>
<td>no-access</td>
<td>The region is unavailable when required roles are missing. All operations including read and write access are denied. All read and write operations on the region will result in <code>RegionAccessException</code> while any required roles are absent. Basic administration of the region is allowed, including <code>close</code> and <code>localDestroyRegion</code>. <em>no-access is the default loss-action only if you define membership roles. Without roles, there is always full access to the region.</em></td>
</tr>
</tbody>
</table>
| reconnect                       | Loss of required roles causes the entire cache to be closed. In addition, this process will disconnect from the distributed system and then reconnect. Attempting to use any existing references to the regions or cache will throw a `CacheClosedException`.  
  
  *This setting requires a full cache configuration in the cache.xml file. All programmatically defined cache and region attributes are lost if they are not also in the .xml.* |
Slow Receivers (Order, Latency)

Another challenge faced by other distributed data management solutions that support for client-side interfaces is data notification of changed data and maintaining data ordering. This problem affects any JSR 107/JCache system such as ehCache, Terracotta, and Coherence; what to do when a client member is slow receiving data? Causes of such conditions include JVM pauses, NIC cards issues, or high data flow to the system such that it cannot keep up. Other systems either disconnect such clients from the system or slow all clients down to the speed of the slowest client. GemFire has an alternative model that dynamically queues data to such clients on a client-by-client basis.

GemFire can be configured to replicate those queues onto multiple computers so that in the event of server failure, clients can be automatically connected to an alternative server that has their queues. This enables clients to continue processing without even being aware of a server failure. To assure that no data being sent to a client is lost, GemFire has a pessimistic model for queue replication. This may result in a small amount of data replay on the client. However, GemFire data processing includes a monotonically increasing message number. So GemFire detects duplicate data and discards it before a client application sees it.

Administrative Functionality

GemFire has a full set of JMX-enabled APIs for management, monitoring and system health. GemFire also has a retrospective analysis capability to help analyze the detailed low-level statistics that GemFire can be configured to produce.

GemFire has a plug-in for Hyperic another product of the vFabric Cloud Application Platform products from VMware. GemFire also includes GFMon to examine the state of a GemFire Distributed System in real time. Below are a few screen shoots of some GFMon capabilities.

A GemFire distributed system produces logs for applications, cache servers and locators.

- **Applications and cache servers** - To create log files, you must set the log-file attribute in the process’s `gemfire.properties` file. Otherwise, the messages go to stdout. These log files can be placed anywhere that is convenient for monitoring. For applications, these log files contain entries from GemFire operation only.

- **Locator** - The locator always creates a log file in its working directory. This logging is not configurable.

Additionally, you can use the JMX Agent to integrate GemFire into any standards based system management tool to perform the following management tasks:

- View the distributed system and its settings
- View distributed system members
- View and modify configuration attributes
- View runtime system and application statistics
- View a cache region and its attributes and statistics
- Monitor the health of a GemFire Enterprise system and its components

GemFire even has a command line interface to handle basic management tasks such starting locators, cache servers, and merging logs. Gfsh - pronounced “g-fish” - is a GemFire command-line tool for browsing and editing data stored in GemFire data fabrics. Its rich set of Unix-flavored commands allows you to easily access data, monitor fabric peers, redirect outputs to files, run batch scripts, execute custom functions, and much more.
Use DataBrowser to browse the data in a GFE cache server by running *ad hoc* OQL queries and to monitor real-time changes to a data region by registering a continuous query.
The administration API allows you to configure, start, and stop a distributed system and many of its components. The API is made up of distributed system administration, component administration, and cache administration. In addition the administration API provides interfaces to issue and handle system member alerts and to monitor statistics. You can receive notifications in an application on all membership events, cache and region creation, and log messages.

The health monitoring API allows you to configure and monitor system health indicators for GemFire distributed systems and their components. There are three levels of health: good health that indicates that all GemFire components are behaving reasonably, okay health that indicates that one or more GemFire components are slightly unhealthy and may need some attention, and poor health that indicates that a GemFire component is unhealthy and needs immediate attention.

Because each GemFire application has its own definition of what it means to be healthy, the metrics that are used to determine health are configurable. GemFire provides methods for configuring the health of the distributed system and members that host Cache instances. Health can be configured on both a global and per-machine basis. GemFire also allows you to configure how often a GemFire health evaluation is conducted. The health administration APIs make it possible to configure performance thresholds for each component type in the distributed system (including the distributed system itself). These threshold settings are compared to system statistics to obtain a report on each component’s health. A component is considered to be in good health if all of the user-specified criteria for that component are satisfied. The other possible health settings, okay and poor, are assigned to a component as fewer of the health criteria are met.
Retrospective Analysis

GemFire has a tool to enable you to examine all of the statistics generated by GemFire and correlate them for retrospective analysis.

In this sample statistics chart in VSD, the manager’s evictions and the virtual GC efforts are good enough to keep heap use very close to the eviction threshold. The eviction threshold could be increased to a setting closer to the critical threshold, allowing the virtual machine to keep more data in tenured memory without the risk of overwhelming the virtual machine. This chart also shows the blocks of times when the manager was running cache evictions.

![Chart](image)

Fig. 10: Analyze GemFire generated statistics with the Visual Statistics Display (VSD) tool.

Network Issues

GemFire offers product-wide port use configuration, which means you can configure every port used by the product. This makes the product easy to manage and more security friendly. In the online product documentation, see the new membership-port-range GemFire property in the chapter, “Configuring the System” in the GemFire Enterprise System Administrator’s Guide. GemFire supports multiple network protocols on a per region basis, including the use of ONLY TCP/IP, or the use of unicast or multicast.

Socket Control

We have yet to find a product that gives you such detailed control over network issues. GemFire provides a number of controls around sockets and network buffers. You can optimize GemFire throughput by having GemFire use a pair of sockets for each thread it generates when communicating with other system members. Or you can have GemFire use a single pair for each member. In the latter case, there will be some context switching between threads to send and receive data to/from other members. You can limit the number of network connections.

IPV 4 vs. IPv6

GemFire is completely IPv6-capable. It also supports mixed IPv4/IPv6 networks, although all system members must use one or the other. Many other products in this space are not yet IPv6-capable.

Disk Issues

Auto-Compaction of Operations Log

To avoid the issues that many DBMSs have where a full transaction log can cause a system halt, the Ops Log in GemFire has automatic compaction. Additionally, it can automatically create rollover versions with the total number of versions preserved and total disk space used, completely configurable.
Handling Catastrophic Loss of Storage Disk Data
GemFire provides a way to start a partitioned cache from disk, even if you cannot recover a missing a disk store. You can revoke it from the system during startup so the other members can start. You revoke a disk store by telling online members that a missing member’s disk store is no longer the most recent.

Online Backup
The GemFire backup creates a backup of disk stores for all members running in the distributed system when the backup command is invoked. The backup works by passing commands to the running system members. Each member with persistent data creates a backup of its own configuration and disk stores. The backup does not block any activities in the distributed system, but it does use resources.

Conclusion
Compared to other commercial or open source offerings, GemFire provides the best data management solution for high performance, low latency and high reliability even on intermittently unavailable network and computer hardware platforms.

Find Out More
For information or to purchase VMware products, call 877-4-VMWARE, visit www.vmware.com, or search online for an authorized reseller. For detailed specifications and requirements, refer to the product documentation.