High Performance Data with VMware vFabric™ GemFire®

Best Practices Guide

October 2011

vmware

© 2011 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. This product is covered by one or more patents listed at http://www.vmware.com/download/patents.html.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc
3401 Hillview Ave
Palo Alto, CA 94304
www.vmware.com

Contents

1. Introduction 5
1.1 Overview 5
1.2 Purpose 5
1.3 Target Audience 5
1.4 Scope 5
1.5 References 6
2. vFabric GemFire Architecture 7
2.1 Overview 7
2.2 vFabric GemFire Topologies 8
3. vFabric GemFire General Administration Guide 18
3.1 Overview 18
3.2 Installation 18
3.3 Configuration 18
3.4 Monitoring 20
3.5 General Administration and Troubleshooting 20
4. vFabric GemFire and Spring 22
5. High Level Tuning 24
5.1 Overview ..... **Error! Bookmark not defined.**
5.2 JVM Memory Segments 24
5.3 JVM Tuning and Best Practices 24
6. vFabric GemFire on VMware Best Practices 29
6.1 Overview 29
6.2 Latency Sensitive Applications Best Practices if Virtualized 29
6.3 Memory Sizing of Virtual Machines Running vFabric GemFire 32
6.4 vCPU sizing of Virtual Machines Running vFabric GemFire 35

© 2011 VMware, Inc. All rights reserved.

© 2011 VMware, Inc. All rights reserved.

1. Introduction

1.1 Overview

This *High Performance Data with VMware vFabric GemFire Best Practices Guide* provides information about best practices for the deployment of data fabric systems. The guide describes the best practices for VMware vFabric™ GemFire® data caching systems and its various design constructs. The document captures four main deployment patterns commonly used to implement enterprise data requirements:

- vFabric GemFire deployed as an enterprise data management system.
- vFabric GemFire deployed as L2 cache.
- vFabric GemFire deployed for HTTP session management.
- vFabric GemFire deployed as a faster mass data mover—for example, real-time reporting.

1.2 Purpose

This guide provides best practice guidelines for deploying vFabric GemFire. The recommendations in this guide are not specific to any particular set of hardware or to the size and scope of any particular implementation. The best practices in this document provide guidance only and do not represent strict design requirements because enterprise data requirements can vary from one implementation to another. However, the guidelines do form a good foundation on which you can build—many of our customers have used these guidelines to successfully implement an enterprise data fiber for their enterprise applications.

1.3 Target Audience

This guide assumes a basic knowledge and understanding of vFabric GemFire, data management concepts, and virtualization with VMware vSphere®.

- Architectural staff can use this document to gain an understanding of how the system works as a whole as they design and implement various components.
- Engineers and administrators can use this document as a catalog of technical capabilities.

1.4 Scope

This guide covers the following topics:

- vFabric GemFire Architecture – This section provides a high level best practice architecture for various topologies that are part of the high performance data solution space.
- vFabric GemFire Best Practices – This section covers various best practices pertaining to setting up a data fabric in production, and GemFire on vSphere best practice considerations.
- vFabric GemFire Monitoring and Troubleshooting Primer – There are times when you have to troubleshoot a particular vFabric GemFire application problem. vFabric GemFire is equipped with several tools such as GemFire Tool for Monitoring (`GFMon`), Visual Statistics Display (`VSD`), and vSphere `esxtop` utilities which are very informative when troubleshooting.
- vFabric GemFire FAQ – In this section, we answer some frequently asked questions about the various data fabric deployments.

© 2011 VMware, Inc. All rights reserved.

1.5 References

It is recommended that you become familiar with the following documentation:

- *vFabric GemFire User's Guide*: https://www.vmware.com/support/pubs/vfabric-gemfire.html
- *Enterprise Java Applications on VMware – Best Practices Guide*
  http://www.vmware.com/resources/techresources/1087
- *Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs*
  http://www.vmware.com/resources/techresources/10220

© 2011 VMware, Inc. All rights reserved.

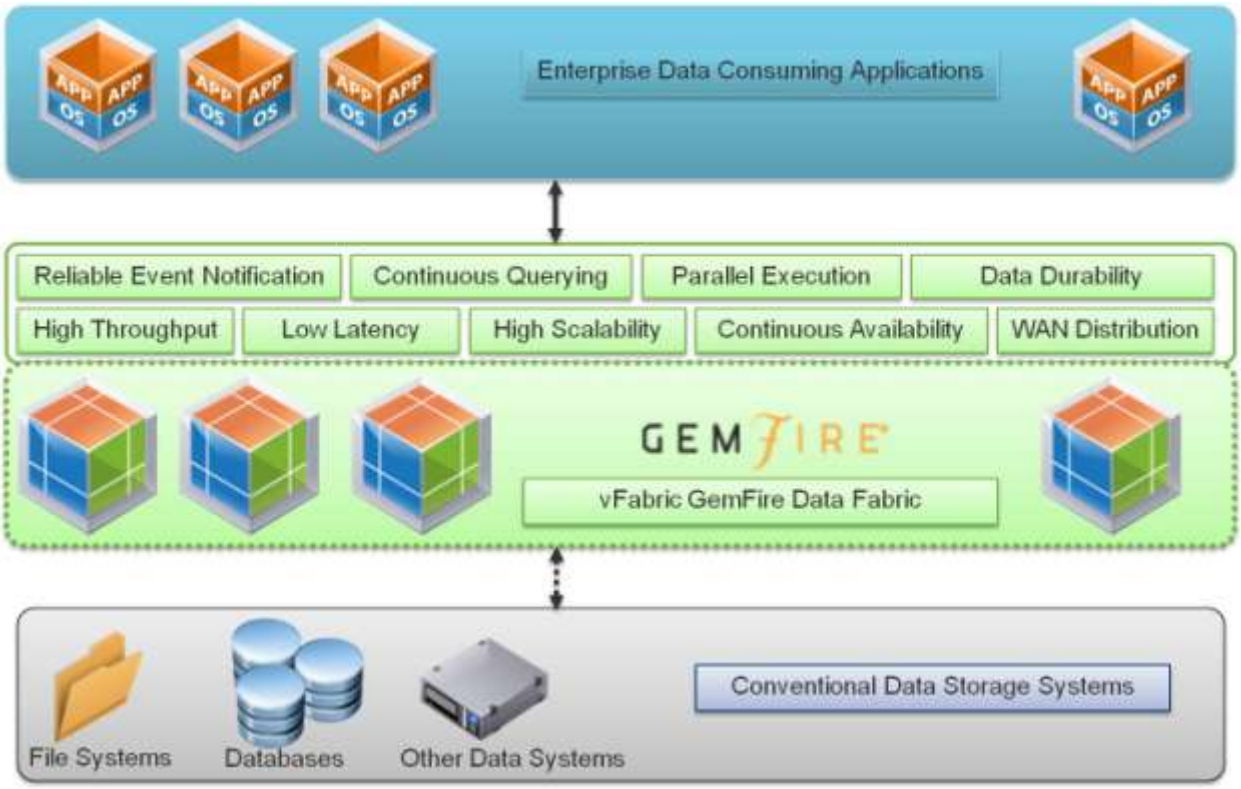
2. vFabric GemFire Architecture

2.1 Overview

vFabric GemFire is an in-memory distributed data management platform that can be spread across many virtual machines, JVMs, and GemFire servers to manage application objects. Using dynamic replication and partitioning it offers the following features in the platform: data durability, reliable event notification, continuous querying, parallel execution, high throughput, low latency, high scalability, continuous availability, and WAN distribution.

The following figure shows GemFire as the middle data tier that orchestrates data delivery from the back-end datastores to the consuming applications. As demand from consuming applications increases, the middle tier data layer expands to appropriately meet demand. For further persistence, resiliency data can be written behind to a backup store like a relational database for archival purposes. GemFire also provides full persistence durability using its own native shared-nothing persistence mechanism.

Figure 1. vFabric GemFire Architecture



Best Practice	Description
BP 1 – Common Distributed Data Platform	When data delivery is required to be at the highest speed possible, if milliseconds and microseconds matter, setting up vFabric GemFire as an enterprise data fabric system is the correct approach. By doing so and as shown in Figure 1, you introduce a common data delivery and consumption layer in-memory for all enterprise applications data needs. This allows you to benefit from the scalability, availability, and speed of execution features of vFabric GemFire.

© 2011 VMware, Inc. All rights reserved.

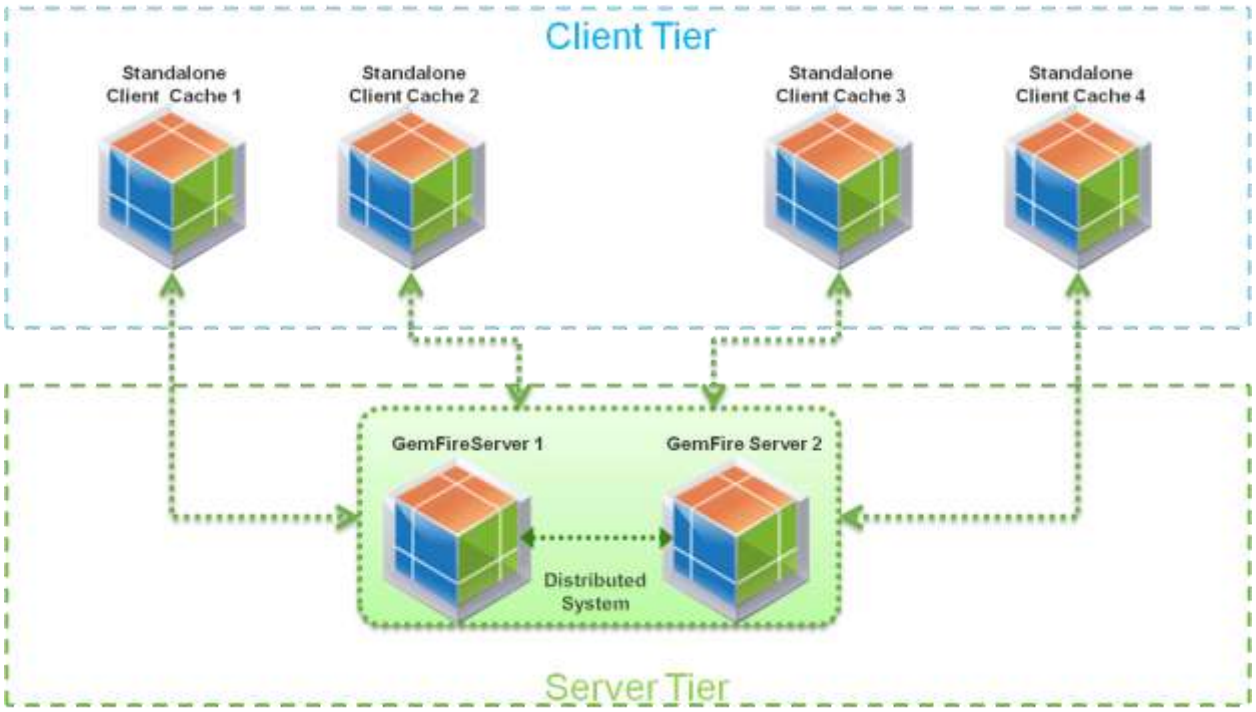
2.2 vFabric GemFire Topologies

There are three main setup topologies for vFabric GemFire: client/server, peer-to-peer, and multisite. Each of these topologies can be used standalone or combined to form an extended, full featured distributed data management system.

2.2.1 Client/Server Topology

In a client/server topology there are two tiers, a *client tier* and a *server tier*. In Figure 1, the client and server tiers are depicted. The client tier communicates with the server tier to search for or update data objects from the server tier. In the client tier, standalone client caches 1, 2, 3, and 4 communicate directly with the server tier.

Figure 2. vFabric GemFire Client/Server Topology



© 2011 VMware, Inc. All rights reserved.

Best Practice	Description

BP 2 – Client/Server Topology	• Client/server topology is the most commonly used for enterprise class applications. The client sends individual operations to the server to update cached data, to satisfy a local cache miss, or to run an ad hoc query. The server streams cache update and continuous query events to the client based on client subscriptions.
	• For advanced tuning and increased throughput capacity, you can distribute the load of network traffic for your client/server traffic through a different adapter than the peer-to-peer traffic by setting a server bind address. These `gemfire.property` lines specify different non-default addresses for the member: `bind-address=10.80.10.80`, `server-bind-address=10.80.10.81`.
	• Use client/server topology when any of the following are requirements of an enterprise class application:
	  ○ Dynamic server discovery – The GemFire server locator utility dynamically tracks server processes and directs clients to new servers, giving clients indirection from explicit server information. Clients need to know only how to connect to the locator services. They do not need to know where servers are running or how many servers are available at any time.
	  ○ Server groups – You can assign your servers to logical groups that your clients can refer to in their connection configurations. For example, you might use groups to manually partition your data, with one group of servers hosting one set of data and another hosting another set. Or you might use a group to direct all database-centric traffic to the subset of servers that are directly connected to a backend database. Servers can belong to multiple groups. Your clients need to specify only the group to use and are isolated from having to know which servers belong to which groups.
	  ○ Server load balancing – The GemFire server locator tracks current load information for all servers, directing new client connections to the servers with the least load. GemFire provides a default load probe for your servers, which you can replace with your own customized plug-in.
	  ○ Server connection conditioning – Client connections can be configured to transparently time out and be replaced with new connections which allows overall server use to be rebalanced after new servers are started. This helps speed conditioning in situations such as adding servers or recovery from server crashes and other downtime.
	  ○ Automated data and query updates – Your clients can subscribe to events in the server. These events can include data updates and updates to results for continuous queries that the client has registered with the server. The server uses subscription queues to send the updates asynchronously.
	  ○ Server failover and high availability – When servers crash, the client connections automatically fail over to the remaining servers. If the servers are sending automated updates to the clients, the update requests also automatically fail over. You can configure redundancy in your server subscription queues so that the failover does not interrupt the stream of events from the server side.

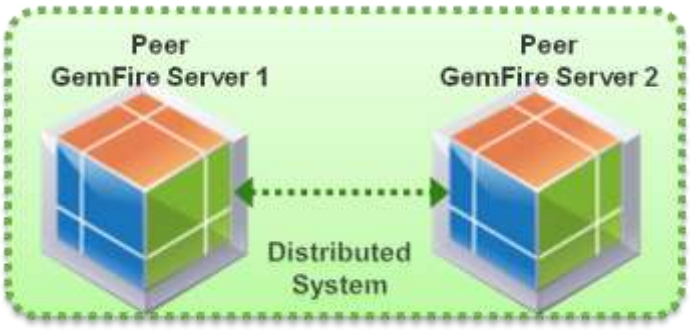
© 2011 VMware, Inc. All rights reserved.

BP 3 – Client/Server Common Sizes	• Because vFabric GemFire is horizontally scalable, scalability is limited only by the hardware resources available. • Configure vFabric GemFire data management nodes to host approximately 1TB of data. • While there is no product limit as to the number of JVMs deployed in a data management system, up to 32 JVMs have been implemented in various production systems. • Thousands of clients can connect back to the data management systems accessing data. Client scalability can be managed through connection pools. • Run within a 64-bit JVM, and a heap size of up to 32GB. When using `-XX:CompressedOops`, the 32GB heap space uses 32-bit pointer addressing which saves large amounts of memory as opposed to using 64-bit pointer addressing. With this approach you can continue to run inside a 64-bit JVM and benefit from larger heap sizes, but with compressed pointer addressing. Any heap size beyond 32GB uses 64-bit pointer addressing because the `-XX:CompressedOops` optimization is limited to 32GB. This is limitation in the Java optimization and is not specific to vFabric GemFire as GemFire supports larger than 32GB heap sizes. **Note** Client local cache generally starts with zero client side data and is enabled only when needed for performance optimization.

2.2.2 Peer-to-Peer

In this topology, two or more intercommunicating vFabric GemFire servers form a distributed system. The data is distributed according to the data region's configuration redundancy rules.

Figure 3. Peer-to-Peer vFabric GemFire Distributed System



© 2011 VMware, Inc. All rights reserved.

Best Practice	Description
BP 4 – Peer-to-Peer Multihomed Machines	If running on multihomed machines, you can specify a non-default network adapter for communication. In non-multicast peer-to-peer situations, communication uses the bind-address property. This address must be the same for all vFabric GemFire servers within the distributed system.
BP 5 – Peer-to-Peer Sockets	• Highly concurrent/high-throughput deployments need `conserve-sockets` set to false and then limit the NIO thread pool servicing clients if (and only if) the number of peer-to-peer sockets/worker threads increases to the point where context switching overhead degrades performance. More nodes in the peer-to-peer cluster imply more connection/worker thread overhead, and thus a possible reason to lower the per-server NIO pool size. Conversely, more powerful hardware—with more available cores, and running on a more powerful underlying network fabric—implies the ability to increase the per-server NIO pool size. • For peer-to-peer threads that do not share sockets, you can use the `socket-lease-time` to limit the time that a socket sits idle. When a socket that belongs to an individual thread remains unused for this time period, the system automatically returns it to the pool. The next time the thread needs a socket, it retrieves one from the pool. • *Socket-buffer-size* determines the buffer size. Buffers should be at least as large as the largest stored objects and their keys, plus some overhead for message headers. The overhead varies depending on who is sending and receiving, but 32KB should be sufficient. Larger socket buffers allow your members to distribute data and events more quickly, but they also take memory away from other requirements. **Note** This provides excellent performance even for small update sizes, while not killing the potential for larger-sized chunking to optimize bulk operations `putAll()`/`getAll()`/queries and rebalancing/failover/failback.
BP 6 – Peer-to-Peer hosts File	Verify that every peer-to-peer host has a `hosts` file entry for itself and for all other hosts on the LAN. The `hosts` file entry format should follow *host* + *domain* (for example, "gemserver1" and "gemserver1.vmware.com") are present for each IP address entered in the hosts file.
BP 7 – Use Locators in Managed Peer-to-Peer Environments	• Locators using TCP/IP – Using this method you run GemFire locator processes that manage the authoritative list of active peer-to-peer distributed system members. These locators are *peer locators*. A new member connects to one of the locators to retrieve the member list which it uses to join the system. Locators are highly recommended for production systems. • For production environments, always use at least two locators on different hosts.

© 2011 VMware, Inc. All rights reserved.

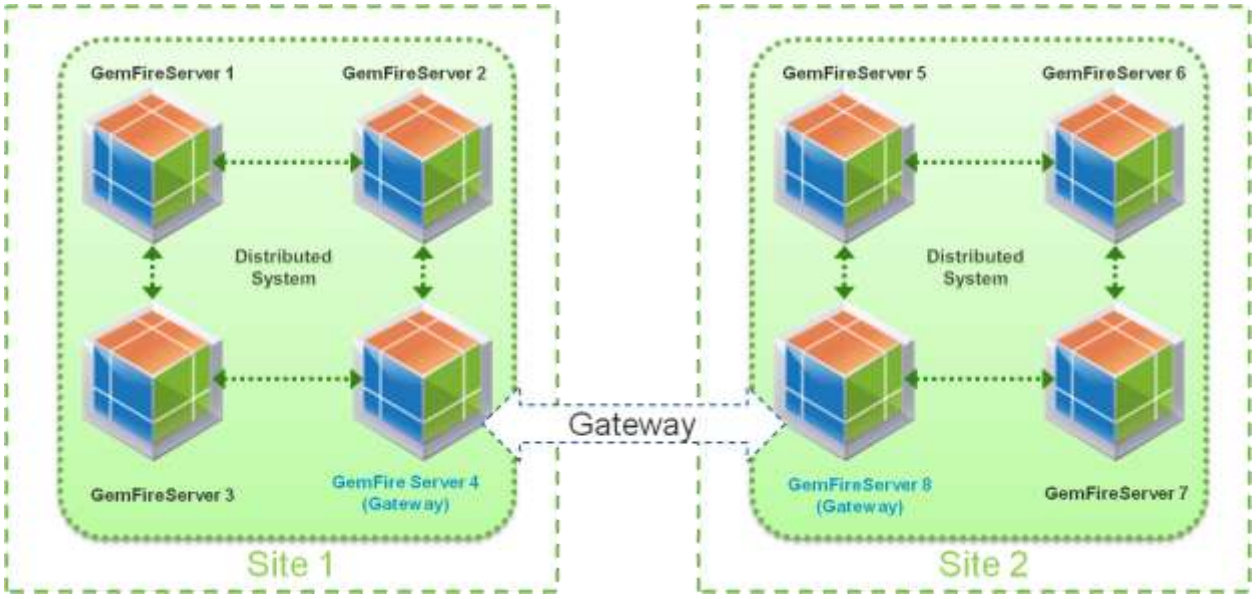
Note The client/server topology is the most commonly used in enterprise applications. There are some rare cases when performance constraints are so strenuous that a single hop is all that can be afforded to meet SLAs. Peer-to-peer topology typically has one network hop between peers as opposed to client/server topology where there are two network hops, if you assume that there are at least two redundant servers with which a client can communicate.

However, when using peer-to-peer topology it is assumed that the rich features of a client/server topology, such as continuous querying, registration of interest, and connectivity through pooled connections are not needed. These features are available only with the client/server topology. Client/server topology is the most commonly used as it is the most feature rich.

2.2.3 Multisite Topology

In the case of multisite topology, as shown in Figure 4, there are two sites each with a distributed system. Within each site, one server member is nominated as the gateway to provide data distribution between sites in case of a failure event, or for other enterprise data distribution requirements. The Site 1 and Site 2 topology can locate both sites within one datacenter, or the sites can be distributed geographically at different datacenters if needed.

Figure 4. vFabric GemFire Multisite Topology



© 2011 VMware, Inc. All rights reserved.

Best Practice	Description

BP 8 – Multisite Topology	• Use multisite topology in distributed data systems that require a robust failover mechanism at the application data layer.
	• Use the conflation feature when using a gateway hub so that only the latest updates are passed over to the remote site. With conflation, earlier entry updates in the queue are dropped in favor of updates sent later in the queue. This is problematic for applications that depend on seeing every update. For example, if any remote gateway has a CacheListener that needs to know about every state change, you should disable conflation. To enable conflation, you can set the `batch-conflation` attribute to true within the `gateway-queue` cache configuration element.
	• In a multisite installation using gateways, messages can back up in the gateway queues if the link between sites is not tuned for optimum throughput. If a receiving queue overflows because of inadequate buffer sizes, it can become out of sync with the sender and the receiver is unaware of the condition. The gateway's <gateway> `socket-buffer-size` attribute should match the gateway hub's <gateway-hub> `socket-buffer-size` attribute for the hubs the gateway connects to. For example:
	`<gateway-hub id="EU" port="33333">`
	`  <gateway id="US" socket-buffer-size="42000">`
	`  <gateway-endpoint id="US-1" host="USHost" port="11111"/>`
	`  <gateway-queue overflow-directory="overflow"`
	`  maximum-queue-memory="50" batch-size="100"`
	`  batch-time-interval="1000"/>`
	`  </gateway>`
	`</gateway-hub>`
	`<gateway-hub id="US" port="11111" socket-buffer-size="42000">`
	`  <gateway id="EU">`
	`  <gateway-endpoint id="EU-1" host="EUHost" port="33333"/>`
	`  <gateway-queue overflow-directory="overflow"`
	`  maximum-queue-memory="50" batch-size="100"`
	`  batch-time-interval="1000"/>`
	`  </gateway>`
	`</gateway-hub>`
	• Avoid overflowing to disk when possible by adjusting the `maximum-queue-memory` attribute to accommodate needed memory. However, should you wish to overflow to disk, you can easily do so to provide additional data reliability.
	• For production systems and higher availability, set `enable-persistence` to true for the `gateway-queue` attribute. This causes the gateway queue to persist to the disk store specified in `disk-store-name`.
	• Although for ease of illustration we show two sites, typically you would implement n+1 sites to achieve fault tolerance.
	• The multisite topology can also span a WAN with multiple sites in, for example, New York, Tokyo, and London. Refer to Figure 5.

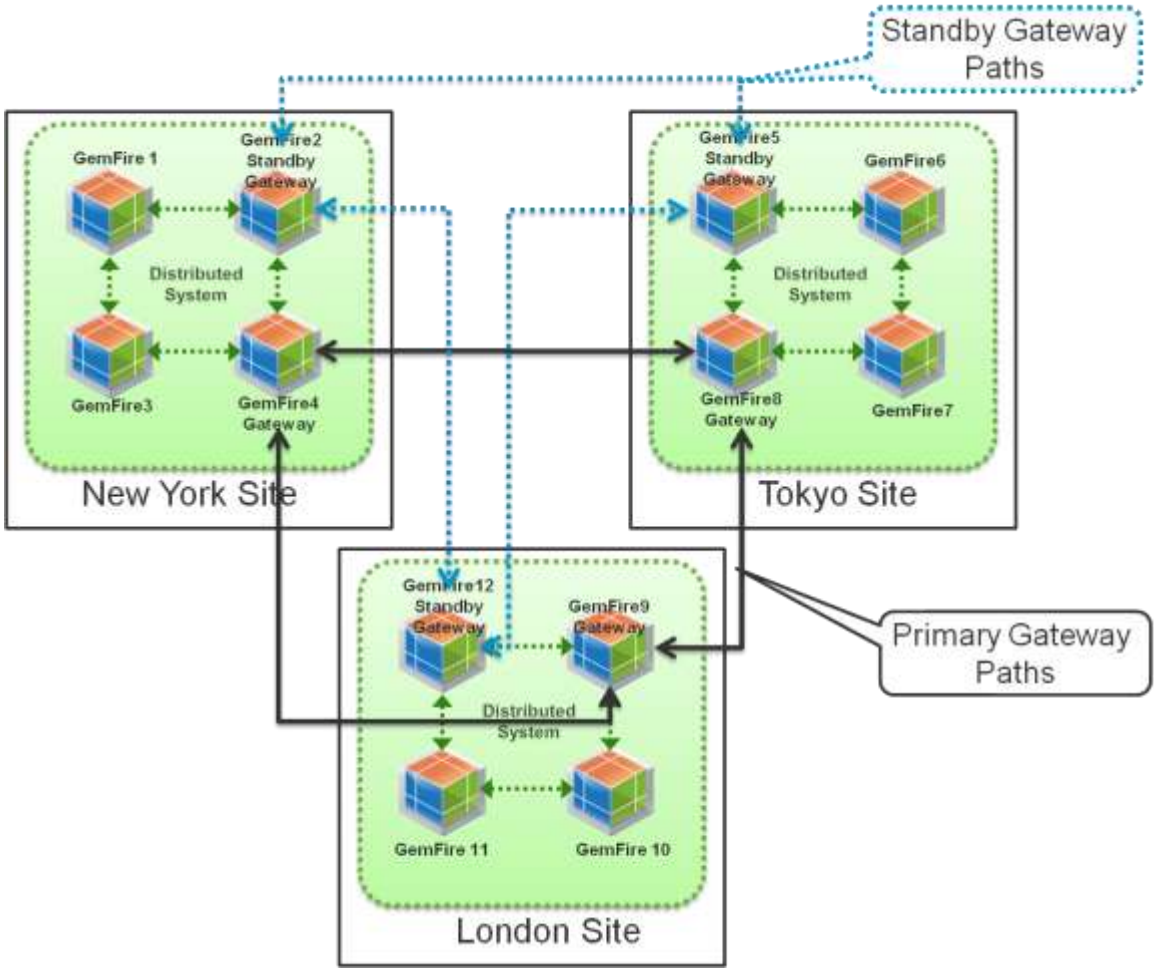
© 2011 VMware, Inc. All rights reserved.

Note Gateway hubs and gateways communicate through TCP/IP sockets. The gateway hub listens at a specified address and port for gateway communication from remote sites. Gateways are configured with endpoint information matching the remote gateway hub specifications. The gateway sends connection requests to the gateway hubs to establish two-way TCP connections. For information on the multisite configuration, refer to the Configuring Multisite Installations section of the *vFabric GemFire User's Guide* (http://www.vmware.com/support/pubs/vfabric-gemfire.html).

In addition to the site-to-site communication, each gateway hub is a member in its own distributed system.

Figure 5 shows three global sites in New York, London, and Tokyo. Each site has a primary gateway and a backup gateway. It is important to inspect and tune the configuration parameters of the WAN gateways.

Figure 5. vFabric GemFire in a Global Multisite Configuration

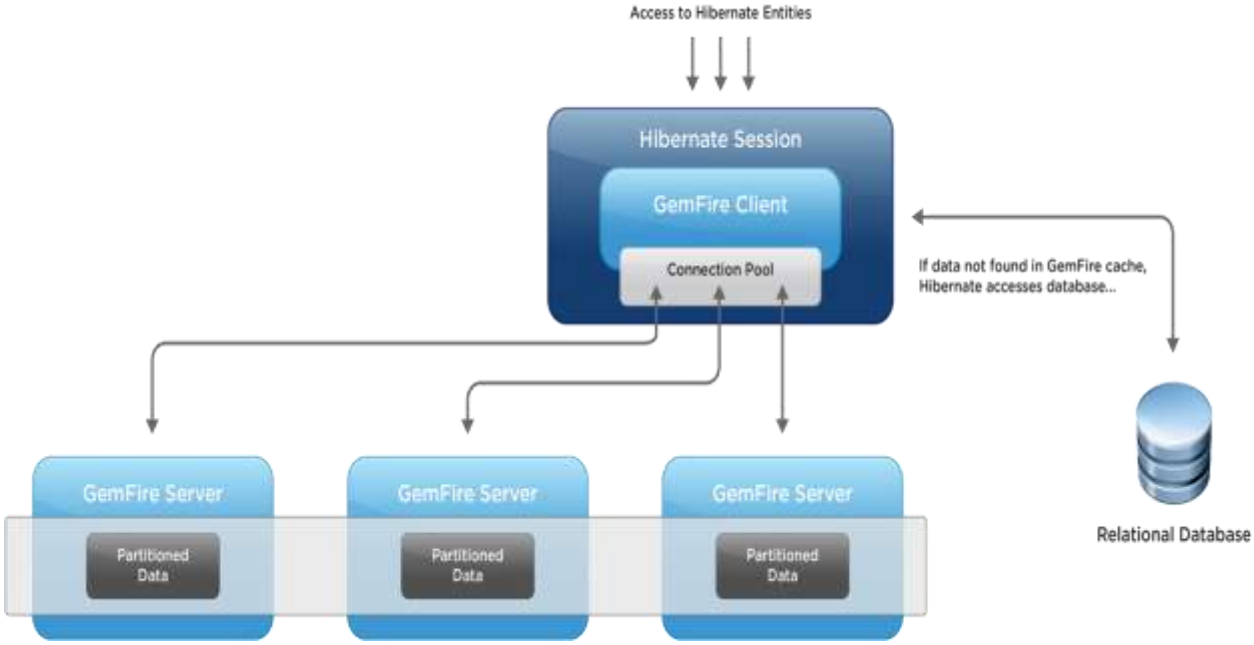


© 2011 VMware, Inc. All rights reserved.

2.2.4 Using vFabric GemFire as Simple L2 Cache

In Figure 6 a client/server topology is used to configure vFabric GemFire as a Hibernate L2 cache. This configuration has the added benefits of faster performance with relative ease of configuration. This is installed as a Hibernate plug-in and therefore no code change is required. It also keeps the query results as distributed cache objects thus improving performance and availability.

Figure 6. Using vFabric GemFire as Hibernate L2 Cache



© 2011 VMware, Inc. All rights reserved.

Best Practice	Description

BP 9 – Hibernate L2 Cache	• Turn on L2 cache in the Hibernate configuration (`hibernate.cfg.xml`):
	`<property name="hibernate.cache.use_second_level_cache">true</property>`
	• Set `region.factory_class` to `GemFireRegionFactory` (`hibernate.cfg.xml` version 3.3+):
	`<property name="hibernate.cache.region.factory_class">`
	`com.gemstone.gemfire.modules.hibernate.GemFireRegionFactory`
	`</property>`
	• Set the cache usage mode to:
	o Read only – Used when you do not plan to modify the data already stored in persistent storage.
	o Read write – Used when you plan to both read from and write to data.
	o Non-strict write – A special read/write mode that has faster write performance. Use this only if no more than one client updates the data at a time.
	o Transactional – Allows for transaction based data access.
	• The cache mode can be set either using annotation or in the Hibernate mapping file:
	o To set using the Hibernate mapping file `entity_name.hbm.xml`:
	`<hibernate-mapping package="PACKAGE">`
	`<class name="ENTITY_NAME" ...>`
	`<cache usage="read-write|nonstrict-read-write|read-only"/>`
	`...`
	`</class>`
	`</hibernate-mapping>`
	o To set the mode using `@Cacheable` and `@Cache annotations`:
	`import org.hibernate.annotations.Cache;`
	`import org.hibernate.annotations.CacheConcurrencyStrategy;`
	`@Entity @Cacheable`
	`@Cache(region = 'REGION_NAME',`
	`usage = CacheConcurrencyStrategy.READ_ONLY|READ_WRITE|NONSTRICT_READ_WRITE|TRANSACTIONAL)`
	`public class MyClass implements Serializable {`
	`...`
	`}`

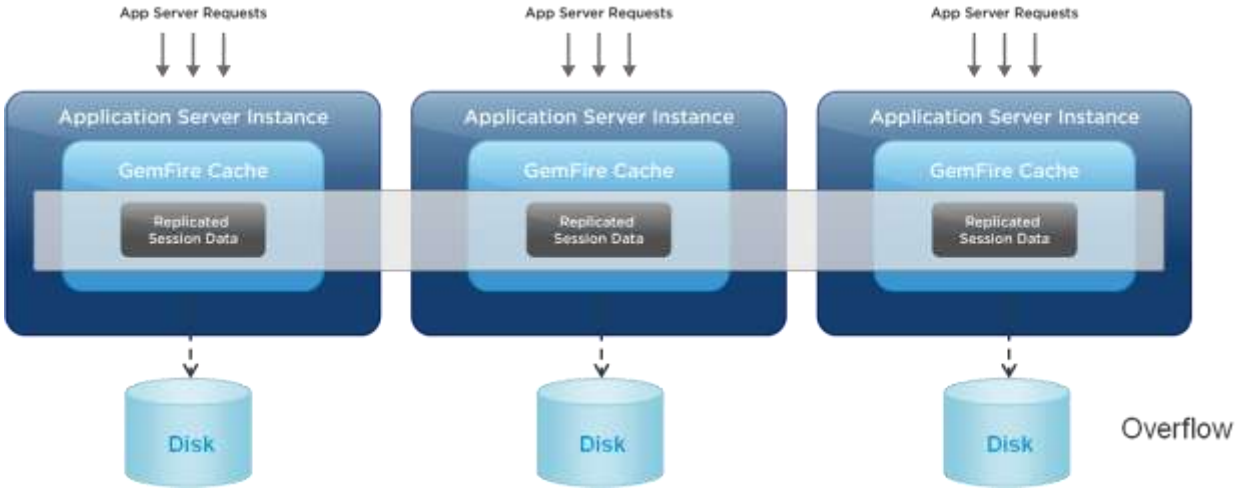
© 2011 VMware, Inc. All rights reserved.

2.2.5 Using vFabric GemFire as an HTTP Session Cache

Best Practice	Description
BP 10 – Topologies for HTTP Session Management	• Either client/server, peer-to-peer, or multisite vFabric GemFire topologies can be used to achieve HTTP session replication. • If dealing with user session data that must be completely fault tolerant, use multisite vFabric GemFire topology and HTTP session management. • Follow the recommended setup in the HTTP Session Management Module section of the *vFabric GemFire User's Guide* (http://www.vmware.com/support/pubs/vfabric-gemfire.html). It is relatively straightforward with minimal change to configure HTTP session replication with GemFire on VMware vFabric™ tc Server™.

In Figure 7, vFabric GemFire is used for HTTP session replication that can be easily achieved when plugged into vFabric tc Server.

Figure 7. Using vFabric GemFire for HTTP Session Replication



2.2.6 Using vFabric GemFire as a Faster Data Mover

Best Practice	Description
BP 11 – Real-Time Reports	• vFabric GemFire client/server topology is most suited for real time report setup. This allows you to move rapidly changing data to the consuming end point client cache to present the data in real time. • vFabric GemFire features such as continuous querying and function execution can help in the implementation of a business critical real time reports.

© 2011 VMware, Inc. All rights reserved.

3. vFabric GemFire General Administration Guide

3.1 Overview

The following sections summarize some high level best practices. There are additional details in the Administration section of the *vFabric GemFire User's Guide*.

3.2 Installation

To download vFabric GemFire, go to http://www.vmware.com/products/vfabric-gemfire/overview.html. Follow the installation instructions at: https://www.vmware.com/support/pubs/vfabric-gemfire.html.

3.3 Configuration

The most notable configuration files within vFabric GemFire are `gemfire.properties`, `gemfireLicenses.zip`, and `cache.xml`.

- `gemfire.properties` – Contains the settings required to join a distributed system. Configuration includes system member discovery, communication parameters, security, logging, and statistics. For a detailed description of the parameters within this file, refer to the *vFabric GemFire User's Guide*.
- `gemfireLicense.zip` – The license file which should never be unzipped.

  Note This is the license file for vFabric GemFire 6.5. With vFabric GemFire 6.6 and later, licensing is done using serial numbers. Refer to the *vFabric GemFire User's Guide* for details.

- `cache.xml` – The declarative cache configuration file. This file contains XML declarations for cache, region, and region entry configuration. It is also used to configure disk stores, database login credentials, server and gateway location information, socket configuration, and so forth.

© 2011 VMware, Inc. All rights reserved.

Best Practice	Description

BP 12 – Configuration	• Do not to unzip the `gemfireLicense.zip` file—leave it intact.
	• Each of the three configuration files has a default name, a set of file search locations, and a system property that can be used to override the defaults. To use the default file specification, place the file at the top level of its directory or jar file. The system properties are standard file specifications that can have absolute or relative pathnames and filenames. If you do not specify an absolute file path and name, the search looks through all the search locations for the file.
	o The `gemfire.properties` file can be specified with the system-level Java property `-DgemfirePropertyFile=<valid file/path>`. You can override any GemFire property set in the file or by the CacheFactory API with a system-level Java argument that follows the pattern `-Dgemfire.<property-name>=<property-value>`.
	• Deploy the same `gemfireLicense.zip` on all members of the peer-to-peer topology, for vFabric GemFire 6.5.
	• For vFabric GemFire 6.6 use the same license key on all members of the peer-to-peer topology.
	• All peer-to-peer members of the distributed system must have the same version of vFabric GemFire. Clients can be up to one major release behind. For example, any 6.x client interoperates with any 6.x or 7.x server, but not with an 8.x server.
	• The vFabric GemFire property `auto-start=true` must be configured for the agents for any version of GemFire 6.5.
	• For highly concurrent workloads, set the GemFire property `conserve-sockets=false` on the data management nodes (DMNs). If the scale is large and too many sockets (and associated threads to service those sockets) are created between the DMNs, tune the CacheServer (a configuration element in the DMN `cache.xml` configuration) to reduce the NIO thread pool servicing client requests. This places a hard upper limit on the possible number of DMN peer-to-peer communication sockets. Refer to the *vFabric GemFire User's Guide* for information:
	`<cache-server max-connections="<integer number>"`
	`max-threads="<integer number>">`
	`</cache-server>`
	• Place the default files either in the current directory from which you start the GemFire server or on the `CLASSPATH`.
	• If you wish to change the default names of these configuration files, you can set the following properties to override them.
	Note These properties are useful to script the deployment or movement of the code base from Dev to QA, and then to production, where there might be a separate set of configuration files for each environment. Depending on which environment is deployed to you can rotate the appropriate files in the `gemfirePropertyFile`, `gemfire.cache-xml-file`, or `gemfire.license-file`.
	• Set `-Djava.net.preferIPv4Stack=true` in the start script for all servers, peers, and locators.

© 2011 VMware, Inc. All rights reserved.

3.4 Monitoring

Best Practice	Description
BP 13 – vFabric GemFire Monitoring Tools	• vFabric GemFire is a specialized product and it is important that administrators are familiar with the available monitoring tools. • The *vFabric GemFire Tools Guide* (http://www.vmware.com/support/pubs/vfabric-gemfire.html) details `GFMon` and Visual Statistics Display (`VSD`) tools available for monitoring. • The `GFMon` tool monitors a vFabric GemFire system in real time, providing health information, detailed operational and configuration data, system alerts, throughput performance, and statistics for system members and connected clients. • The `VSD` tool reads GemFire statistics and produces a graphical display for analysis. Configure the vFabric GemFire property `statistics-enabled=true` to generate statistics files that can be viewed with `VSD`. This can be critical to troubleshoot potential problem areas or help to diagnose a problem. • You can also use the VMware vFabric™ Hyperic® GemFire plug-in that provides a live data user interface for viewing metrics in real time. Refer to the *vFabric Hyperic Guide*: http://pubs.vmware.com/vfabric5/index.jsp • You can also use `esxtop` to monitor vSphere, refer to the troubleshooting section of the *Enterprise Java Applications on VMware – Best Practices Guide* http://www.vmware.com/resources/techresources/1087.

3.5 General Administration and Troubleshooting

Best Practice	Description
BP 14 – General Administration	• For managing disk stores, security, system logs, troubleshooting, the Command Line Utility, and for administering the Distributed System, refer to the *vFabric GemFire User's Guide.* • Do not configure firewall restrictions between the LAN hosts. There must be least four dedicated ports open through any intervening firewall (connecting to the peer-to-peer hosts): one for the locator port, one for the server port, one for the agent HTTP port, and one for the agent RMI port. • If troubleshooting a connectivity problem, set `log-level=fine` on all sides of the connection. This should always include the locator, as it is the first point of contact for connectivity. At a fine log level, you can immediately see whether a socket connection is made, and if it is, why and where the connection is rejected. After troubleshooting it is important to revert to the default log level, or typically in production, to the `log-level=error`. • When you first begin to diagnose a potential connectivity or general problem with the system, start with `telnet` to test whether a remote locator, server, or agent port is reachable.

© 2011 VMware, Inc. All rights reserved.

BP 15 – General Trouble-shooting	The *vFabric GemFire User's Guide* has a detailed section on Troubleshooting and System Recovery. Follow those instructions.

BP 16 – Trouble-shooting SYN Cookies	• When troubleshooting performance problems, check to see you are not impacted by SYN cookies. *SYN cookies* are the key element of a technique used to guard against SYN flood attacks. Daniel J. Bernstein, the technique's primary inventor, defines SYN cookies as "particular choices of initial TCP sequence numbers by TCP servers." In particular, the use of SYN cookies allows a server to avoid dropping connections when the SYN queue fills up. Instead, the server behaves as if the SYN queue had been enlarged. The server sends back the appropriate SYN+ACK response to the client but discards the SYN queue entry. If the server then receives a subsequent ACK response from the client, the server is able to reconstruct the SYN queue entry using information encoded in the TCP sequence number.
	• To check for the presence of SYN cookies:
	`grep SYN /var/log/messages Aug  2 12:19:06 w1-vFabric-g1 kernel: possible SYN flooding on port 53340.`
	`Sending cookies.`
	`Aug  2 12:54:38 w1-vFabric-g1 kernel: possible SYN flooding on port 54157.`
	`Sending cookies.`
	`Aug  3 10:46:38 w1-vFabric-g1 kernel: possible SYN flooding on port 34327.`
	`Sending cookies.`
	• To determine whether or not SYN cookies are enabled (1 is on, 0 is off):
	`$ cat /proc/sys/net/ipv4/tcp_syncookies`
	`1`
	• To temporarily disable SYN cookies (changes at reboot):
	`# echo 0 > /proc/sys/net/ipv4/tcp_syncookies`
	• Permanently disable SYN cookies:
	`Add\modify the following in /etc/sysctl.conf`
	`# Controls the use of TCP syncookies`
	`net.ipv4.tcp_syncookies = 0`

© 2011 VMware, Inc. All rights reserved.

4. vFabric GemFire and Spring

Best Practice	Description

BP 17 – vFabric GemFire and Spring	• Use Spring to configure GemFire servers and regions rather than manually creating them with application code.
	  ○ This allows you to centralize your application service configuration, as opposed to having the Spring context configuration plus a separate `cache.xml` file.
	  ○ You can eliminate the need to implement Declarable on your cache loaders, cache listeners and cache writers. Spring takes care of binding these components into GemFire regions through dependency injection. These components can also be shared among regions as singletons using Spring's normal DI techniques.
	  ○ You can leverage advanced techniques to configure GemFire via SpEL, selectively exposed configuration parameters, and so on.
	• Use the Spring GemFire project to move your GemFire configuration into the Spring context more easily.
	  ○ Refer to *Spring GemFire Home* at http://www.springsource.org/spring-gemfire.
	  ○ Use the GemFire schema extension to the Spring context configuration to simplify the configuration of the various GemFire components further with validated configuration property names (Section 1.1 in the Spring GemFire documentation).
	  ○ Use `GemfireTemplate` to simplify interactions with the GemFire APIs.
	  ○ `GemfireTemplate` includes many best practice techniques for dealing with resource management and multiple threads in a virtual machine working with GemFire.
	  ○ `GemfireTemplate` eliminates the need for you to deal with checked exceptions.
	  ○ The `GemfireTemplate` provides the best practice technique for ensuring thread safe access to GemFire resources in a single virtual machine.
	  ○ `GemfireTemplate` provides utility methods to access and manage data more simply.
	• Transaction management using Spring and Spring GemFire:
	  ○ This provides a portable, well integrated way to provide transactions to your application to promote well defined behavior for multiple threads modifying the data.
	  ○ Automatically configures the GemFire server with best practice recommendations for safest use in a multithreaded environment by enabling `copyOnRead` to prevent client threads from inadvertently editing data contents in a non-transactional way.
	• Use `InstantiatorFactoryBean` to automatically generate an efficient Instantiator:
	  ○ Reflection is the default technique used to serialize and deserialize data across the entire distributed data management system.
	  ○ If data serialization is a bottleneck in your application, the general best practice recommendation is to implement custom instantiator logic to speed up the serialization process.
	  ○ The `InstantiatorFactoryBean` takes a list of domain types (and a unique integer ID for each type to efficiently serialize type info as an integer instead of a string) and automatically generates instantiators using the ASM bytecode library that prevents having to use reflection to serialize and deserialize object data.

© 2011 VMware, Inc. All rights reserved.

With transactions, you must use care if operating in an environment where you might access different types of transactional resources. This is true even if using JMS and JDBC together in a single Spring application and they must be coordinated into a single transaction. Spring gives you patterns that you can use to resolve this.

One pattern is simply not to coordinate at all. You define multiple transaction managers, and then choose the appropriate transaction manager that should apply to a particular set of code (either with `@Transactional`, or programmatic transaction). In this way, the code that updates GemFire directly would use the GemFire transaction manager, and the code that works with some other resource would use the appropriate transaction manager for that resource.

Another pattern is to coordinate those two resources into a single JTA transaction. This requires either a JTA transaction manager like JOTM or Atomikos, or you would have to run your application in a Java EE container. You could then enlist all your transactional resources into a single JTA transaction manager, and not have to worry about coordinating them. The GemFire use of optimistic transactions should help reduce the chance for deadlocks. Also, using `GemfireTemplate` should help reduce this further.

If using a write-through approach with a `CacheWriter` writing to a database or some other transactional resource, then it is highly recommended to use a JTA transaction manager to direct that GemFire's updates and any other resource are automatically part of the same transaction.

With write-behind style updates of a database, this is not as much of an issue because the write to the database is occurring in a different thread from the cache updates. Write-behind is typically more common due to the performance increase it provides.

© 2011 VMware, Inc. All rights reserved.

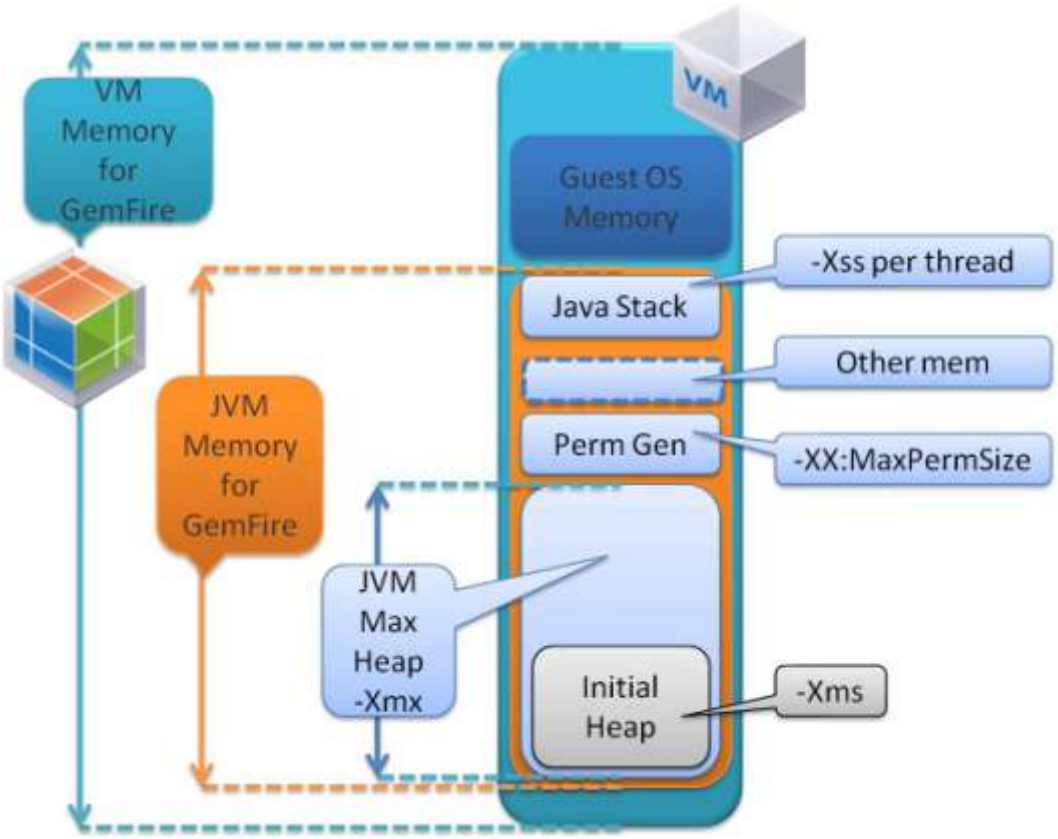
5. High Level Tuning

Systems using vFabric GemFire as an enterprise data management system, where speed and consistency of data is of critical to the operations of the business, can benefit from the tuning guidelines in this section.

5.1 JVM Memory Segments

In Figure 8, the various memory segments are shown in the configuration of running one vFabric GemFire on one JVM and one virtual machine.

Figure 8. JVM Memory Segments of One JVM Running on One Virtual Machine



VM Memory for GemFire = Guest OS Memory + JVM Memory for GemFire

JVM Memory for GemFire = JVM Max Heap (`-Xmx` *value) + JVM Perm Size (*`-XX:MaxPermSize`*) +*

*NumberOfConcurrentThreads * (*`-Xss`*) + "other mem"*

© 2011 VMware, Inc. All rights reserved.

Best Practice	Description
BP 18 – JVM Version	Use JDK 1.6.0_24 or later.
BP 19 – Use Server Flag	Always add the `-server` flag to the JVM command that starts the GemFire server. On server class hardware most JVMs default to this, but it is a best practice to explicitly turn it on.
BP 20 – Set Initial Heap Equal to Maximum Heap	Set `-Xms` (*initial heap*) to `-Xmx` (*maximum heap*). **Note** Without this setting, performance can suffer if the initial heap setting is not adequate and the JVM responds by increasing the memory allocated, causing overhead at runtime.
BP 21 – Garbage Collector Choice	Set `-XX:+UseConcMarkSweepGC` to use the concurrent low-pause garbage collector and the parallel young generation collector. The low-pause collector sacrifices some throughput to minimize stop-the-world GC pauses for tenured collections. It requires more headroom in the heap, so increase the heap size to compensate.
BP 22 – Disable Calls to System.gc()	Set `-XX:+DisableExplicitGC` to disable full garbage collection. This causes calls to `System.gc()` to be ignored, avoiding the associated long latencies.
BP 23 – Occupancy Fraction and Tenured Generation	Set `-XX:CMSInitiatingOccupancyFraction=50` or even lower for high throughput latency-sensitive applications that generate large amounts of garbage in the tenured generation, such as those that have high rates of updates, deletes, or evictions. This setting tells the concurrent collector to start a collection when tenured occupancy is at the given percentage. With the default setting, a high rate of tenured garbage creation can outpace the collector and result in OutOfMemoryError. Too low of a setting can affect throughput by doing unnecessary collections, so test to determine the best setting.

© 2011 VMware, Inc. All rights reserved.

Best Practice	Description
BP 24 – New Generation Size	• Set the `-Xmn` value to be large enough to avoid the new generation filling up. Making the new generation large enough avoids the cost of copying objects into tenured space which can impact performance. • A common approach is to set the `-Xmn` size to approximately 33% of the heap's maximum size, that is, 33% of `-Xmx` for heap sizes less than 8GB. For heap sizes from 8GB–100GB, the 33% rule might be too high—typically 10%-15% of the maximum heap size for these is adequate. To establish the best size for your application you must load test and measure how often the new generation fills up. Based on this, decide whether to adjust the `-Xmn` value. • Prescribed range for smaller heaps: if the heap is less than 8GB, `-Xmn` < 33% of `-Xmx`. • Prescribed range for larger heaps: if the heap is much greater than 8GB, then `-Xmn` < 10% to 15% of `-Xmx`. However, it is difficult to apply this rule across the board on varied workload behaviors. For example, with a very large heap, much greater than 8GB, it is often found that instead of using a percentage calculation, you can choose to cap it at 2GB. Then you can progressively make adjustments based on how quickly you see the young generation fill-up and how it is impacting performance. • A further rule to consider when sizing `-Xmn` is that typically partitioned regions most likely have higher numbers of short lived objects and therefore can require a larger `-Xmn` value. Compare this with replicated regions where the rate of change is minimal and not many short lived objects are created, and thus a lower `-Xmn` is adequate. **Note** This guidance has a caveat that depends on the behavior of the application. For example, if the application requires many Query calls, configure a size at the upper end of the above prescribed `-Xmn` range. For mostly Put calls, configure a size in the middle of the range.
BP 25 – Using 32-Bit Addressing in a 64-Bit JVM	When memory is constrained, set the `-XX:CompressedOops` JVM option. This uses a 32-bit pointer address space within a 64-bit JVM for heap sizes up to 32GB. This can save substantial memory space, in some cases up to 50%, although the savings varies based on the Java types used within the application. **Note** Although this best practice saves on memory space usage, it can lower the throughput and increase latency.

© 2011 VMware, Inc. All rights reserved.

Best Practice	Description
BP 26 – Stack Size	In most cases the default `-Xss` stack size is too large, ranging in size from 256KB–1MB depending on the JVM and operating system. Reduce the stack size to conserve the memory in use by the Java process. For example, you can set `-Xss192k`. Although you can experiment to determine what is best for your environment, you might reduce this to a point where a stack overflow occurs, in which case you have reduced it too far. At that point you should increase it to the point where the stack overflow exception no longer occurs during the load test.
BP 27 – Perm Size	It is a common best practice to set `-XX:MaxPermSize` in the range from 128MB–256MB, although the actual size can vary for your application, and appropriate testing should be conducted. The `PermSize` is where class level information is kept. In the HotSpot JVM this is allocated outside the heap, that is, in addition to the `-Xmx`.

BP 28 – Region Placements in a JVM	• Place both replicated and partitioned regions within one JVM instance; that is, within one data management node. This provides the best scalability and performance. By default, GemFire partitions each data point into a bucket using a hashing policy on the key. The physical location of the key-value pair is abstracted away from the application. Additionally, there is no default relation between data stored in different partitioned regions. To run transactions on partitioned regions, you must colocate all data accessed in any single transaction on the same data host. Additionally, in many cases, you can get better performance if you colocate similar data within a region and between regions. For example:
	  ○ A query run on a patient, her health records, and her insurance and billing information is more efficient if all of the data is grouped in a single JVM.
	  ○ A financial risk analytical application runs more quickly if all trades, risk sensitivities, and reference data associated with a single instrument are located together.
	• Colocation generally improves the performance of data-intensive operations. You can reduce network hops for iterative operations on related datasets. You can usually significantly increase overall throughput for compute-heavy, data-intensive applications. You specify colocation through configuration attributes and custom coding. You can:
	  ○ Add custom partitioning code to a region to route logically connected data into the same buckets.
	  ○ Colocate multiple regions, so buckets with the same number in each region are stored in the same JVM.
	• You can override the default vFabric GemFire mechanism to provide your own colocation. Refer to information about PartitionResolver in the *vFabric GemFire User's Guide*.

© 2011 VMware, Inc. All rights reserved.

BP 29 – vFabric GemFire Development and API Best Practices	The vFabric GemFire product is rich in features. Application developers must be familiar with the following topics that are documented in the *vFabric GemFire User's Guide*. • Quick Start Guide • GemFire Member • Data Regions and Operations • Region Management • Client/Server Configuration • Handling Events • Delta Propagation • Querying and Indexing • Continuous Querying • GemFire Transactions • Function Execution • Managing Member Relationships • `Cache.xml`

© 2011 VMware, Inc. All rights reserved.

6. vFabric GemFire on VMware Best Practices

6.1 Overview

vFabric GemFire and vSphere provide a robust complement to deliver data faster and more reliably using cost effective x86 commodity hardware and vSphere. If virtualizing vFabric GemFire, you can leverage the best practices discussed in the *Enterprise Java Applications on VMware – Best Practices Guide.*

Best Practice	Description
BP 30 – *Enterprise Java Applications on VMware – Best Practices Guide*	The best practices discussed in this guide also apply to vFabric GemFire. Refer to the *Enterprise Java Applications on VMware – Best Practices Guide* http://www.vmware.com/resources/techresources/1087.

6.2 Latency Sensitive Applications Best Practices if Virtualized

There are certain Guest OS, and network considerations that especially pertain to virtualized workloads such as vFabric GemFire. These can benefit from the following best practices.

Best Practice	Description
BP 31 – Guest OS	Red Hat Enterprise Linux 5 and prior versions incur higher virtualization overhead due to high frequency timer interrupts, frequent access to virtual PCI devices for interrupt handling, and an inefficient Linux timekeeping mechanism. By selecting a more current version of Linux, such as SUSE Linux Enterprise Server 11 SP1 or Red Hat Enterprise Linux 6 based on 2.6.32 Linux kernels, or Windows Server 2008, these causes of virtualization overhead are minimized. In particular, Red Hat Enterprise Linux 6 has a tickless kernel that does not rely on a high frequency interrupt-based timer, and is therefore much friendlier to virtualized latency-sensitive workloads. Refer to *Timekeeping best practices for Linux guests* (http://kb.vmware.com/kb/1006427), and *Timekeeping in VMware Virtual Machines* (http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf).

© 2011 VMware, Inc. All rights reserved.

BP 32 – Physical NIC

- Most 1GbE or 10GbE Network Interface Cards (NICs) support a feature called interrupt moderation or interrupt throttling, which coalesces interrupts from the NIC to the host so that the host does not spend all its CPU cycles processing interrupts. However, for latency-sensitive workloads, the time that the NIC delays the delivery of an interrupt for a received packet or for a packet that has successfully been sent on the wire, is time adding to the latency of the workload.
- Most NICs also provide a mechanism, usually with the `ethtool` command, to disable interrupt coalescing. VMware recommends to disable physical NIC interrupt moderation on the VMware ESXi™ host as follows:

  ```
  # ethtool -C vmnicX rx-usecs 0 rx-frames 1 rx-usecs-irq 0 rx-frames-irq 0
  ```

  Where `vmnicX` is the physical NIC as reported by the ESXi command:

  ```
  # esxcli network nic list
  ```

  You can verify that your settings have taken effect by issuing the command:

  ```
  # ethtool -c vmnicX
  ```
- Note that although disabling interrupt moderation on physical NICs is extremely helpful in reducing latency for latency-sensitive virtual machines, it can lead to some performance penalties for other virtual machines on the ESXi host, as well as higher CPU utilization to deal with the higher rate of interrupts from the physical NIC.
- Disabling physical NIC interrupt moderation can also defeat the benefits of Large Receive Offloads (LRO), because some physical NICs (such as Intel 10GbE NICs) that support LRO in hardware automatically disable it when interrupt moderation is disabled, and the ESXi implementation of software LRO has fewer packets to coalesce into larger packets on every interrupt. LRO is an important offload for driving high throughput for large message transfers at reduced CPU cost, so this trade off should be considered carefully. Additional details are available in the knowledge base article, *Poor TCP performance can occur in Linux virtual machines with LRO enabled*: **kb**.vmware.com/**kb/1027511**.
- If the ESX host is restarted the above configurations must be reapplied.

© 2011 VMware, Inc. All rights reserved.

BP 33 – Virtual NIC

- ESXi virtual machines can be configured to have one of the following types of virtual NICs: Vlance, VMXNET, Flexible, E1000, VMXNET2 (Enhanced) or VMXNET3. This is described in *Choosing a network adapter for your virtual machine* (http://kb.vmware.com/kb/1001805).
- Use VMXNET3 virtual NICs for your latency-sensitive or otherwise performance critical virtual machines. It is the latest generation of paravirtualized NICs designed for performance, and is not related to VMXNET or VMXNET2 in any way. It offers several advanced features including multiqueue support, Receive Side Scaling, IPv4/IPv6 offloads, and MSI/MSI-X interrupt delivery. Modern enterprise Linux distributions based on 2.6.32 or newer kernels, like Red Hat Enterprise Linux 6 and SUSE Linux Enterprise Server 11 SP1, ship with built-in support for VMXNET3 NICs, so it is unnecessary to install VMware Tools to get VMXNET3 drivers for these guest operating systems.
- VMXNET3 by default also supports an adaptive interrupt coalescing algorithm, for the same reasons that physical NICs implement interrupt coalescing. This virtual interrupt coalescing helps drive high throughputs to virtual machines with multiple vCPUs with parallelized workloads (for example, multiple threads), while also striving to minimize latency of virtual interrupt delivery.
- However, if the workload is extremely sensitive to latency, then VMware recommends that you disable virtual interrupt coalescing for VMXNET3 virtual NICs as follows.
  - Use VMware Programmatic APIs to add the special virtual machine configuration options as defined in *VMware vSphere Web Services SDK Documentation* (http://www.vmware.com/support/developer/vc-sdk/). Refer to the *VMware vSphere API Reference Documentation*, under the `VirtualMachine` Managed Object Type, for the `OptionValue[] extraConfig` property of the `VirtualMachineConfigInfo` configuration property.
  - To do so manually, first power off the virtual machine. Edit your virtual machine's `.vmx` configuration file and locate the entries for VMXNET3, as follows:

    ```
    ethernetX.virtualDev = "vmxnet3"
    ethernetX.coalescingScheme = "disabled"
    ```

  - Power on the virtual machines for the virtual interrupt coalescing settings to take effect.

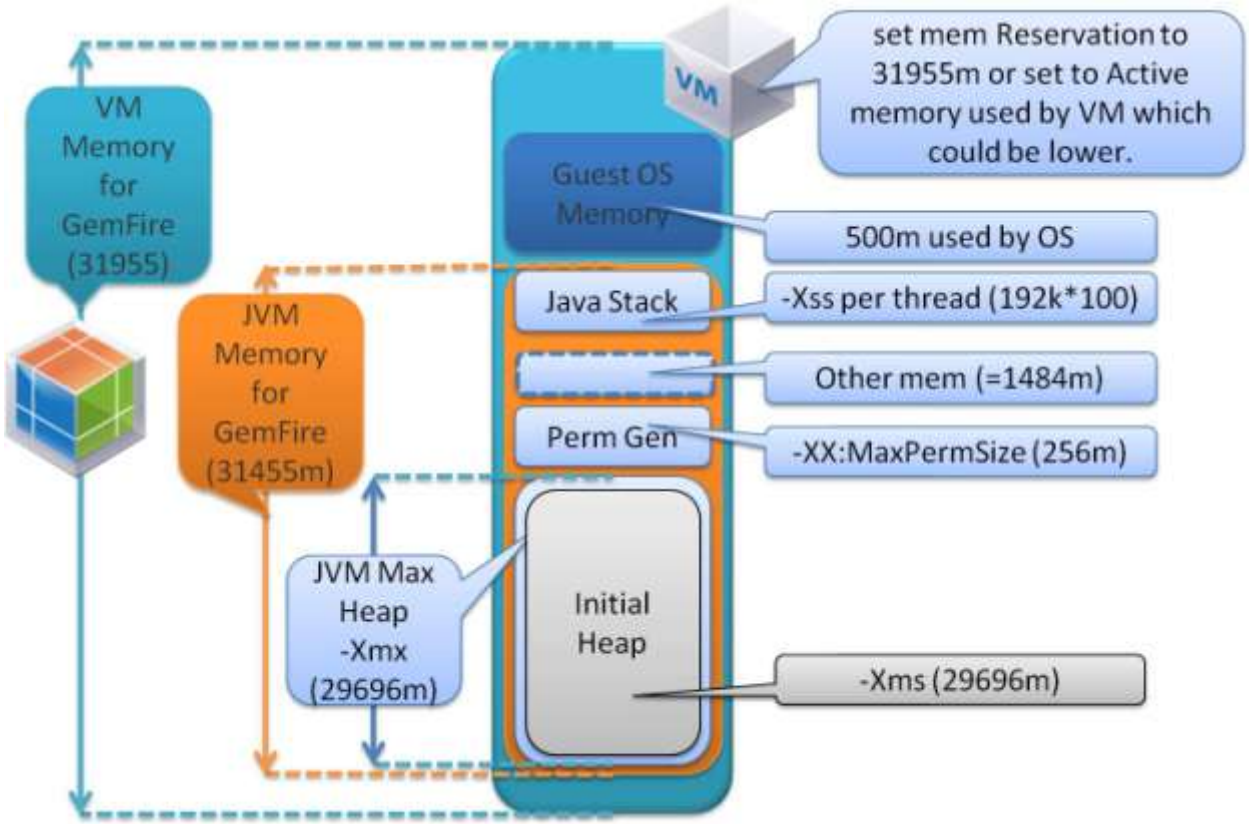
  Note This new configuration option is available only in ESXi 5.0.

© 2011 VMware, Inc. All rights reserved.

6.3 Memory Sizing of Virtual Machines Running vFabric GemFire

As described in *Enterprise Java Applications on VMware – Best Practices Guide* (http://www.vmware.com/resources/techresources/1087), setting an appropriate memory reservation is essential for Java based workloads. Unique to GemFire workloads, it is always advisable to have at least 50% headroom for optimal operation. Refer to Figure 9.

Figure 9. Virtual Machine Sizing Example: One GemFire Server Running on One JVM and One Virtual Machine



VM Memory for GemFire = Guest OS Memory + JVM Memory for GemFire

JVM Memory for GemFire = JVM Max Heap (-Xmx *value) + JVM Perm Size (*-XX*:MaxPermSize) + NumberOfConcurrentThreads * (*-Xss*) + "other mem"*

© 2011 VMware, Inc. All rights reserved.

Using the VM memory elements of the equation in Figure 9 with the following settings, the sizing assumptions are as follows,:

Where:

- *Guest OS Memory* is approximately 0.5GB-1GB (depends on OS/other processes).
- `-Xmx` is the JVM max heap size.
- `-Xss` is the Java thread stack size. The default is OS and JVM dependent, and can range from 256k-to 1MB.
- *Perm Size* is an area additional to the `-Xmx` (*Max Heap*) value and is not GC-ed because it contains class-level information.
- *"other mem"* is additional memory required for NIO buffers, JIT code cache, classloaders, Socket Buffers (receive/send), JNI, GC internal info.
- *VM Memory for GemFire = Guest OS memory + JVM Memory for GemFire*
- Let's assume that through load testing a JVM max heap (`-Xmx`) of 29696m has been determined as necessary. This max heap is made of actual memory usage of region data that was determined to be 19797m, and applying the best practice of adding 50% head room, this translates to 19797 actual data usage + 10000m head room, which implies a total of 29696m. Therefore, `-Xmx` would be set to 29696m.

Proceed to size as follows:

- Set `-Xmx29696m` and `-Xms29696m`.
- Set `-XX:MaxPermSize=256m`, which is a common number and depends on the memory footprint of the Class level information within your Java application code base.
- The other segment of *NumberOfConcurrentThreads**(`-Xss`) depends largely on the *NumberOfConcurrentThreads* the JVM will process, and the `-Xss` value you have chosen. A common range of `-Xss` is 128k-192k.
    - `-Xss` is OS and JVM dependent. If the stack is not sized correctly you will get a StackOverflow. The default value is sometimes quite large and you can benefit from sizing it down to help save on memory consumption.
    - If for example *NumberOfConcurrentThreads is 100, then 100*192k => 19.2m* (assuming you set `-Xss` to 192k)
- Assume the OS has a requirement of about 500MB to run as per the OS spec.
- Total JVM memory (Java process memory) = 29696m (`-Xmx`) + 256m (`–XX:MaxPermSize`) + 100*192k (*NumberOfConcurrentThreads**`-Xss`) + "other mem." Therefore JVM memory approximately equals 29696m+256m+19.2m + *"other mem"* = 29971m + *"other mem"*
    - Typically, *"other mem"* is not significant. However, it can be quite large if the application uses lots of NIO buffers and socket buffers. This value can be approximated with ample space as about 5% of the heap. That is, 5% * 29696= 1484m, although proper load testing should be conducted to verify.
    - This implies that JVM process memory = 29971m+1484m=31455m.
- To determine the VM memory, assume you are using Linux with no other significant process running on it (only this single Java process), the total configured memory for the virtual machine translates to:

    VM memory for GemFire Server = 31955MB + 500MB = 31955m

© 2011 VMware, Inc. All rights reserved.

- Set the VM memory as the *memory reservation*. You can choose to set the memory reservation as 31955m, but over time you should monitor the active memory used by the virtual machine that houses this JVM process and adjust the memory reservation to that active memory value, which could be less than 31955m.
- This also means that the NUMA rules apply and you want to make sure that each socket on the server has at least 32GB of RAM to house this virtual machine, along with the vCPUs needed.

Best Practice	Description
BP 34 – Set Memory Reservation and Allow for 50% of Memory Headroom	• If your GemFire region data requires a GemFire server to have 19797m of memory, allow for 50% headroom for optimal performance. This implies 19797m + 10000m (50% overhead), therefore a total of 29696m. • Following the sizing example shown in Figure 9, set the memory reservation to approximately 31955m. Setting a memory reservation directs that the reserved physical memory is made available by VMware ESX® or ESXi to the virtual machine when it starts. • Do not overcommit memory. • When sizing memory for vFabric GemFire server within one JVM on one virtual machine, the total reserved memory for the virtual machine should not exceed what is available within one NUMA node for optimal performance. Refer to BP 35 for further NUMA discussion and considerations.

© 2011 VMware, Inc. All rights reserved.

6.4 vCPU sizing of Virtual Machines Running vFabric GemFire

Best Practice	Description
BP 35 – Enable Hyper-Threading and Do Not Overcommit CPU	• Always enable hyper-theading. • Do not overcommit CPU as vFabric GemFire are typically latency-sensitive applications that are CPU-bound. Size these based on the available physical cores. • For most production vFabric GemFire servers, size with a minimum of two vCPU virtual machines. However, larger 8-vCPU virtual machines might be necessary in some cases to achieve your SLAs. • Encourage good NUMA locality by sizing virtual machines to fit within the NUMA node. If you suspect your virtual machine has poor NUMA locality, inspect the **N%L** counter from `esxtop`. With good NUMA locality, this counter should be 100%. **Note** A NUMA node is equivalent to one CPU socket, so for a server with two sockets there are two NUMA nodes. Therefore, the available number of physical CPU cores and RAM is divided equally among the NUMA nodes. This is critical when sizing virtual machines to fit within one NUMA node. ○ For example, a 2-socket 16-core (8 cores on each socket) server with 192GB RAM, has two NUMA nodes, each with 8 physical cores (CPUs) and 96GB RAM (192/2). When sizing virtual machines it is important not to exceed the limits of 8 vCPUs and 96GB RAM. Exceeding any of these CPU and RAM maximums of each NUMA node can force the virtual machine to fetch memory from a remote location, impacting performance. There are many ESX/ESXi CPU scheduler enhancements to avoid this, but following this example can help. ○ You can set `sched.cpu.vsmpConsolidate = "true"`, as described in the knowledge base article, *Consolidating vCPUs for an SMP virtual machine can improve performance for some workloads* (http://kb.vmware.com/kb/1017936). This instructs the ESX scheduler to place the vCPUs of an SMP virtual machine into the fewest Last Level Cache (LLC) possible. This policy benefits by providing better cache sharing.
BP 36 – Cache Server, JVM and VM Ratio	• Have one JVM instance per virtual machine. Typically, this is not a requirement. However, because vFabric GemFire JVMs can be quite large (up to 100GB), it is advisable to adhere to this rule in this case. • Increasing the heap space to service more data demand is better than installing a second instance of a JVM on a single virtual machine. If increasing the JVM heap size is not an option, then consider placing the second JVM on a separate newly created virtual machine, thus promoting more effective horizontal scalability. As you increase the number of GemFire servers, also increase the number of virtual machines to maintain a 1:1:1 ratio among the GemFire server, the JVM, and the virtual machines. • Size for a minimum of two vCPU virtual machines with one GemFire server running in one JVM instance. This allows ample CPU cycles for the garbage collector, and the rest for user transactions.

© 2011 VMware, Inc. All rights reserved.

BP 37 – VM Placement	Because vFabric GemFire can place redundant copies of cached data on any virtual machine, it is possible to inadvertently place two redundant data copies on the same ESX/ESXi host. This is not optimal if a host fails. To create a more robust configuration, use VM1-to-VM2 anti-affinity rules to indicate to vSphere that VM1 and VM2 can never be placed on the same host because they hold redundant data copies.
BP 38 – vMotion, DRS Cluster, and GemFire Server	• When you first commission the data management system, place VMware Distributed Resource Scheduler (DRS) in manual mode to prevent an automatic VMware vSphere® vMotion® migration that can impact response times. • vMotion can complement vFabric GemFire features during scheduled maintenance to help minimize downtime impact due to hardware and software upgrades. It is a best practice to trigger vMotion migrations over a 10GbE network interface to speed up the vMotion process. • Do not allow vMotion operations with vFabric GemFire locator processes as the latency introduced to this process can cause members of the vFabric GemFire servers to falsely suspect that other members are dead. • Use DRS clusters dedicated to vFabric GemFire. If this is not an option and GemFire has to run in a shared DRS cluster make sure that DRS rules are set up that will not use vMotion to migrate vFabric GemFire virtual machines. **Note** In some cases a vMotion migration might not succeed and instead fails back due to a rapidly changing volatile memory space, which can be the case with Partitioned regions and in some cases of Replicated regions. The failback is a fail-safe mechanism to the source virtual machine and it does not impact the source virtual machine. vMotion makes this failback decision based on the time it takes to complete the iterative copy process that captures changes between the source virtual machine to the destination virtual machine. If the changes are too rapid and vMotion is not able to complete the iterative copy within the default 100 seconds, it checks whether it can failsafe to the running source virtual machine without interruption. Therefore, vMotion only transfers the source virtual machine to the destination if it is certain that it can complete the memory copy.
BP 39 - VMware HA and vFabric GemFire	• VMware HA should be disabled on vFabric GemFire virtual machines. If this is a dedicated vFabric GemFire DRS cluster, you can disable HA across the cluster. However, if this is a shared cluster, it is important to exclude vFabric GemFire virtual machines from HA. **Note** Set up anti affinity rules between the vFabric GemFire virtual machines that will not cause any two GemFire servers to run on the same ESX host within the DRS cluster.

© 2011 VMware, Inc. All rights reserved.