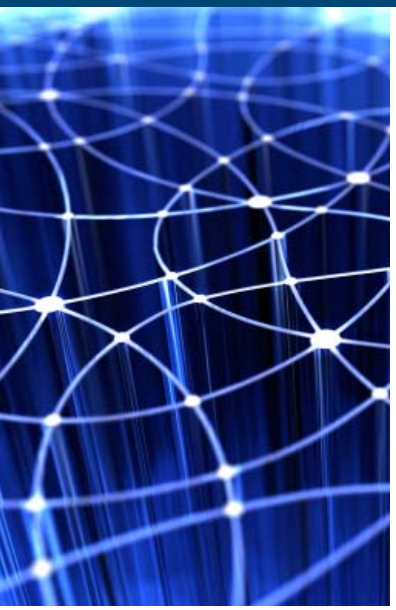

GEMFIRE®
the enterprise
data fabric

High Performance, Mission Critical Applications for the war-fighter:

Solutions to network challenges and today's fluid combat environment

There are key challenges in building mission critical applications that must work in the face of the following conditions:

- Data access when data sources (especially multiple) are not collocated with or near the application (increasing latency)
- Times/situations when the network or data sources or both are unavailable
- Low bandwidth and/or unreliable networks
- Data that is changing at high rates and these changes need to be made visible to multiple applications rapidly
- Automatic data synchronization between applications and data sources when the application is cut off from the network and then returns to the network
- Hardware, software, network failures
- Combine data from multiple data sources and create efficient data placement
    - o move data by interest
    - o rollup data from multiple sources
    - o drill down to details on as needed

To enable agile, robust, dependable information, it must be available when it is needed at speeds that enable it to matter. With Net-Centric systems, the network is the database. To avoid having the network become the problem, data must be delivered so as to meet these goals. Recently accessed data should be available even if the network is off-line. Data should be as current as the network allows.

Information is a force multiplier. Accurate and timely information is a bigger force multiplier. With the advent of GIG-BE, war-fighters have come to expect fast and reliable access to information in garrison and command centers. Unfortunately, network access erodes as war-fighters move to more mobile environments, ships, mobile command centers, special-forces with sat radio, etc. Mobile environments have significantly degraded capabilities to deliver information affecting timely information access. It can be frustrating and lethal not to be able to access information that was available only minutes before when a network connection vanishes. The ability to subscribe to and receive continuous updates at a fine grained level to key information can enable planners and operators to get the details that can cause them to change their minds as the situation changes. With today's much more fluid combat environments, high level commanders need access to more tactical details than ever before. And they need those details immediately.

If it was only possible to hang onto information that was being looked at minutes before, or to browse information in garrison or on the network and have it travel with the war-fighter as they deployed. If only it was possible for that information to be automatically updated whenever that person was on the network. If only it was possible to subscribe at a fine grained level to key information and get alerted as data that affected an operation changed, operators could be aware of the details that planners depended on changed. And if only it was possible to take tactical real-time feeds, to aggregate them and roll them up to high levels, yet have ability to drill down to the detailed information if necessary.

The simplest use case is to maintain access to critical data in the face of network outages. For applications such as the NCES Mediation Service or the IA Security service applications, simple data caching can enable access to data that they have been working on even though the network connection is down. Services would only look in the data cache if they were unable to reach the source data over the network. This could be because the network is down or the service is down or the data source is down.

The next level of capability is to reduce the latency and network bandwidth usage. Simple caching is insufficient to solve this issue for data that has high rates of change or for which changes have serious impact to the application, because the cache may be out of sync with the actual data source. However, if a service or data source actively send changes to the cache(s) (real-time updates when network is available and guaranteed sequential delivery of data change events after a network outage) then it is possible to:

- Get data in face of network outages
- Reduce network band usage
- Speed data access

If the cache supports notification of data changes back to the application, then it is also possible for applications to be aware of data changes to the data they have previously retrieved (i.e. do automatic view maintenance). Most applications today are not written to expect this level of service. But it can have a significant benefit to be aware that data in the application is stale.

This still doesn't solve the issues of low bandwidth network and high data change rates. So the next level of capability would be a caching technology that supports those capabilities necessary to only send data of interest to applications, so that unnecessary bandwidth is not spent sending data that is not yet of import to the user. For example, the GCCS-J TMS application is able to manage 100,000 tracks with a 1 minute update rate (which works out to approximately 1,650 update per second of 1k objects or 1 MB per second). With thousands of concurrent users that totals Gigabytes per second. Most users can only look at thousands of object of interest (and zoom down to even less when analyzing a situation such as air defense cover of the straits of Tehran). So by being able to filter the data and detect the delta, bandwidth usage can be reduced an order of magnitude or more. Additionally, if many application users sit at a location, then the ability to tier software so that local server caches can be shared by many local user caches can also significantly reduce bandwidth usage. In this case it can also reduce latency for the 2nd, 3rd and nth local user of that data. For example, in the case of a carrier at sea, sending the same video clip, or thumbnail image to multiple users, results in that data traveling over a very finite pipe more times than is necessary. Ideally, a solution that handled these issues would allow for multiple expressions of fine grained interest, so that if users were interested in multiple subsets of data that had different characteristics they could get them all. e.g. Enemy air tracks and friendly air defense units. Any solution to handle these issues must be capable of scaling to handle any arbitrary rate of data change (so long as the necessary network to send the data is available).

Interest in data should be able to last for as long as necessary and should not actually require a human being sitting at a screen. Imagine a mission planning system that retrieves data from a number of sources such as tracks, order of battle, and battle damage assessment. In the course of planning the mission, new information could arise, a target gets destroyed, a new enemy SAM unit shows up, which would cause different decisions during planning. Even after the mission started, knowledge of changes to the underlying data could be distributed to participants that needed it. Once data from multiple sources with fine grained detail can be pulled into an application, new questions can be asked of the data not anticipated by the designer of either source. For example, merge MIDB with 'contains' information about equipment with track data and ask for all current tracks of EW capable aircraft.

Next are the issues of synchronization when applications that were cut off from the network and are reconnected at a later time. Data synchronization falls into two categories, easy (in that it does not have to know about application context), and hard (application context matters because of potential conflicting updates when the application was off-line). Although the "easy" data synchronization problems involve lots of complex technology to automate the synchronization process, they do not require any work from application developers and are thus deemed easy. The "hard" data problems require only a little technology, but require a lot of domain knowledge and work from application developers, and are thus deemed hard. (At least for the people actually building an application).

Where multiple users can not update the same data in a distributed environment, the servers need to queue data changes to individual clients based on their interests. This may seem a simple challenge but for fast changing data this can create a queue of significant size that exceeds the client's and network's ability to ever have the client catch up. In that case a mechanism that conflates changes so that the client only sees the most recent update to each piece of data would minimize the overall size of the queue and maximize the chance of success. If message ordering is to be preserved for all clients, then queuing of messages is also necessary for client applications that fall behind in their ability to process incoming data because they are too busy. If a client "never" recovers for any of the above reasons, then it may be desirable for performance reasons to be able to automatically remove their queues and data interests until they reemerge. By "never", we really mean that a client falls more than some configurable amount of time or number of messages behind the overall system.

Where multiple users can update the same data, the caching mechanism must have a pluggable way of resolving conflicts. Since a simple last one to update the data "wins" does not work for most applications, application specific code that checks originators, timestamps, version numbers, or other application specific data, must be able to be applied to incoming data to apply correct changes and deny incorrect changes to data. Notification of failed updates because of data conflicts must be propagated back to the original application instance that tried to update the data. A mechanism that validates incoming data can also be used to merge, roll up, or otherwise combine multiple data instances in the data fabric into a single element. This too would depend on application specific code that is invoked as data arrives.

Because this paper is focused on describing potential solutions to challenges for mission critical applications, it is important to consider failure conditions. So far all of the potential failures described have been network issues or edge client issues. Preventing data loss and assuring continuity of operations in the event of server hardware loss is also critical. To assure data integrity across multiple instances of the same application, messages to clients must be delivered even in the event of a server loss. To keep up with data rates for high speed data, this often must be done without committing data to disk. It is critical to replicate 'in memory' the server caches and queues of messages

waiting to be sent to clients that have fallen behind or are off the network. This data should be able to be replicated on N nodes, (where N is the number needed for the application to reach the desired level of redundancy to assure continuous operation). Having done this, it is possible to have client applications fail-over seamlessly from one server to another in the event of server loss. Such a mechanism can even accommodate site failure, although depending on data rates, clients may need to resynchronize with the servers. The ability to fail-back as servers become available is also critical to maintain load balanced and guarantee SLAs.
If distributed caching is used to manage the current operational data. Some high bandwidth locations, the Command HQ's, the Pentagon, etc, could combine data from multiple servers and roll that data up into a larger operational picture. Only the higher level rolled-up data need be sent up the command chain. Since the details still exist at each of the individual servers where the data came from, and since the data can be queried in such a way that deltas are continuously streamed back to an application, then the larger, more coarse grained operational picture has the ability to enable commanders to drill down for live details when they need them. This enables a virtual total picture to be available without saturating the available bandwidth.

In order to have the ability to solve the challenges of high data rate mission critical applications that must work in the face of various network issues, be able to filter data, allow for occasionally connected applications, handle synchronization, be tolerant of hardware failures, and scale up or down as needed, a new class of software infrastructure is needed. This software must be able to provide for all of the capabilities described in this paper including dynamic data filtering, high data rates, handling slow receivers, handling disconnected users, managing network or hardware loss, preventing data loss, have mechanisms to ensure data integrity and the ability to scale to data rates as needed, in massively distributed environments. Such a type of software, a "Data Fabric" is available today from GemStone Systems. It is being used by 7 of the top ten Wall Street investment banks for program and algorithmic trading, risk and data analysis, global order book management (across WAN connections), global market and reference data distribution of original and derived data to applications and user around the globe. It has been put into a next generation GCCS-J/COP prototype.

GemFire is a high performance, distributed data management infrastructure. It provides distributed in-memory data caching, application level notification of data changes, configurable ACID properties, query, and enterprise scale. It supports multiple topologies including tiered servers, with edge clients. It supports fault tolerance for hardware and network failures at either the server or client level. This enables support for disconnected operations. Its memory management abilities enable it to be configured to replicate data for high concurrent loads, and for high availability. Data in memory can be grouped into separate Regions to support data placement and to support separating data from multiple applications. The distributed memory management also provides for methods to gracefully expand capacity to meet scalability and performance goals. GemFire has been benchmarked at over 100,000 1k inserts per second while distributing data to 1,000 client VMs. In general it is limited only by network bandwidth and available hardware. In order to provide for the ability to maximize performance and also support the level of concurrency needed for individual distributed applications, GemFire supports both high coherence and loose coherence across the distributed memory space. This included support for distributed transactions in a peer-to-peer environment.

The ability of applications to register interest in data with fine-grained queries, using OQL, a superset of SQL 92, from the Object Data Management Group (ODMG), enables applications and users to dynamically have the data they need move to clients from one or more servers. Applications receive a result set from the query. Then applications continue to receive notification as data changes on the server change the query results. Updates, deletes, and inserts are sent to the client result set to provide automatic view maintenance. Client applications receive a call-back and can take appropriate actions in response to the data changes. This ability like all of the data delivery mechanisms in GemFire is fault tolerant. Servers have the ability to automatically queue data for clients that are slow receiving data or are currently off the network. Data and queues can be replicated across multiple servers. If a server dies, clients seamlessly switch to an alternate server and continue receiving data from where they left off.

GemFire also provides for the ability to run regular queries that do not continue to send data changes to the application. Regular queries can be written to bring back less than the full object. For large objects, such as Air Tasking Orders, which can exceed 40 MB, this ability can significantly increase performance and reduce network bandwidth, when users are only interested in a small subset of the data.

GemFire provides for data persistence. This combined with the continuous query capabilities enable the easy creation of client applications that can be shutdown, moved, restarted, and periodically brought on the network to send or receive data changes. For the issues of conflict resolution that emerges when data can be updated at multiple locations GemFire provides multiple mechanisms to plug in application defined code to handle conflict resolution. There are two major ways to handle this issue. The first is to have updates travel via a separate GemFire Region to a process that is used to scrub data inserts. For the GCCS-J TMS-E application prototype, this method is used as all user changes, pass back into the correlation engine. The other methods is to have server-side plug-ins, called cache-writers,

compare the input data to the current state of the data, and then decide which data state is correct, using time-stamps, authoritative source, version numbers, or other application specific data. If this method needs additional reference data to make the correct decision, this data could easily be stored inside GemFire in another Region. Since client application block when inserting data into GemFire, changes that are rejected are immediately visible to end-users. Note, this method only works when users are on-line. For off-line changes, data must flow to an alternate region where data can be examined using the first method.

GemFire also provides the enterprise scale management capabilities needed deploy large groups of distributed caches. It is highly instrumented. Its capabilities include distributed management, monitoring, alerting, and administration of the distributed system. All of these capabilities are exposed through the industry standard management API's JMX. It also provides for retrospective analysis of performance. It is highly configurable and tunable over with over 100 configuration parameters.

Given NCES's vision statement to "enable the secure, agile, robust, dependable, interoperable data-sharing environment for DOD where war fighter, business, and intelligence users share knowledge on a global network", a distributed caching capability that provides for disconnected operations, high speed data distribution, notification of data changes, and enterprise level fault tolerance and management could provide significant capabilities to further this vision.

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

Regional Sales Offices:
New York | 5 Penn Plaza, 23rd Floor New York, NY 10001 | Phone: 646.530.8458
Washington D.C. | Phone: 301.564.0550

GEMSTONE®