JMX Federation

Requirements

Mandatory Requirements

1. Single Agent View. User should be able to monitor and manage entire distributed system from single window - for example Jconsole
2. JMX Remote Interface for each Member. Each Member can be accessed remotely using JConsole or any other tool over RMI.
3. Efficient SerializationAll traffic between members should be handled in a such way to avoid in-efficient serialization.
4. Scalability. As number of member grow impact of monitoring should not increase.
5. Integration with Gemfire Security Authentication and Authorization Framework
6. Interoperability with non-java tools Can any non-java tool monitor Gemfire easily? (HTML Adaptor and REST Interface?)
7. Linear overhead of Monitoring. As members grow impact should not follow non-linear path.

Other Desirable Requirements

1. Efficient Polling Support. Polling is very generic usecase of monitoring. hence should be efficiently handled either by using push updates from members or using bulk requests and cashing responses untill next poll interval
2. Push updates from managed members to managing members to reduce network traffic and calls.
3. Dynamic and Intelligent system to optimize collection/polling based on history or request set based on current hot-set of MBeans, Managed mbeans and managing mbean can work together to optimize the polling.
4. Bulk Request/Response. If request set is known ahead of time(prediction) entire set can be requested in one single request so as to avoid multiple polls.
5. Non JMX Extensions
    1. Bulk Mbeans which accept multiple request and return data in bulk
    2. Streaming of VSD
    3. Streaming of Logs

Assumptions

Approaches

Federation

Window to Distributed system is going to be single node which will federate incoming request to all managed nodes. There are multiple approaches to achieve this requirement.

JMX Remote RMI Connector

This approach involves having proxy connection of each mbean server located in every managed member.

Pros

1. Simple to implement and standard compliant
2. Reliable and performant
3. Part of JDK

Cons

1. Scalability : Each remote mbean has proxy in managing node. No coarse grained connections.
2. One more remoting system along side Gemfire.
3. Costly in terms of serialization of Composite Data
4. Communication model is JMX so very finely granular(mbean-level). No direct way of doing bulk-request response.
5. For "Aggregate" scope Mbeans we have to take care of the threading, timeouts, and pooling. (As plan is to gather information from members concurrently)

Gemfire Connector

This approach involves using Gemfire P2P system for getting data and invoking operations. The communication mode uses generic request-response model similar to JSON. (See **Appendix** 1 for samples snippets)

Pros

1. Complete control over message strcture which gives opportunutiy for efficient design (less-granular, bulk-requests etc.)
2. Bulk-Requste-Response possible
3. Effficient serialization for composite and complex data.
4. Members and exachnge lot of other metadata to reduce monitoring overhead like current hot-set of mbeans requested by JMXClient etc.
5. For "Aggregate" scope Mbeans inbuilt mechanism can be used for threading, timeouts and pooling.

Cons

1. Version Dependency. If gemfire version changes and part of M&M change we need to do extra round of testing for JMX Federation.
2. P2P Threads overhead. It will interfere with gemfire data requests. "**Thread-owned sockets**" pattern from Wiki Page can help us to separate the resources into Admin and Data requests helping us to avoid competition between two.

Hybrid Approach

This apporach involves using gemfire p2p request-response model on top of direct RMI connection.

This approach combines advantages of gemfire P2P while avoiding dis-advantages of RMI and gemfire p2p approach.

Pros

1. Reliability and stability of RMI
2. All benefits of gemfire p2p like bulk requests, serialization benefits

Cons

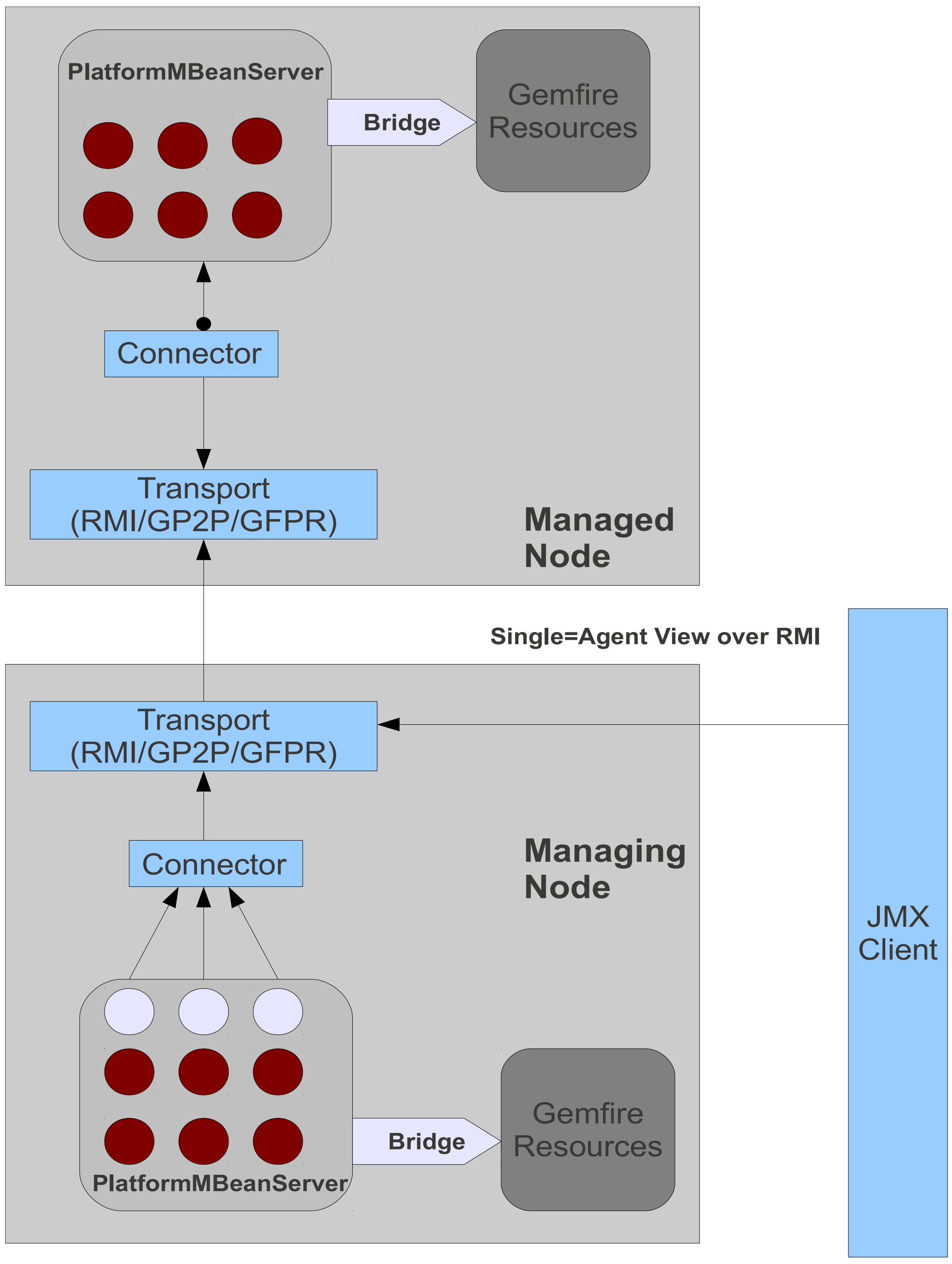
1. One more remoting system side by side of Gemfire

Comparison of Approaches

Feature	**RMI**	**Gemfire**	**Hybrid**
Message Model	No Control over message model. You can only define Composite Attributes	Generic Request/Response Model. Full control	Generic Request/Response Model. Full control
Coarse grained Request	No. At best is All attributes from on Mbean(getAttributes Method)	Multi-attribute requests over multiple mbean across multiple members	Multi-attribute requests over multiple mbean across multiple members
Multi-attribute Requests (Bulk Requests)	No Directly supported by JMX Standards but possible using SuperMBean	Yes	Yes
Aggreage Requests(Across Members)	No. Aggregation need to implement	Yes	Yes
Overhead on Gemfire Data requests/Separate Channel of Communication	Clear separation of resources	No separation. With separate processType it can be separated to some extend	No separation. With separate processType it can be separated to some extend
Serialization of Composite Data	No Control.	Gemfire DataSerizlizable is used	Gemfire DataSerizlizable is used

Architectural/Design Description

Block Diagram



Terms

- ***Managed Node*** : Gemfire distributed system members
- ***Managing Node*** : Gemfire distributed system member who acts as proxy for external JMX client and forwards request to approapriate managed members
- ***Bridge*** : It is component which extracts all monitoring related information from Gemfire

Distributed system in its VM. It is abstraction component to encapsulate gemfire APIs from plain Mbeans. It also has responsibilty of invoking mbean operations on gemfire distributed system components

- ***Transport*** : this is the main component which will federate requests to managed members. Currently considered approaches are RMI, GP2P : Gemfire P2P, and gemfire fixed partitioned regions. RMI Connector will always be present on Managing node.
- ***Connector*** : This is abstraction which understand Mbean ObjectName and other JMX objects and transforms it for appropriate format that underlying transport understands.
- ***Local Mbeans*** (Red coloured small circles) : Use bridge for getting information on local cache .
- ***Proxy Mbeans*** (White coloured small circles) : Use connectors for getting information on remote nodes
- ***JMX Client*** : Any generic client like Gfsh, Data-browser, Gfmon like Dashboard, Jconsole.

Detailed Description

The two approaches discussed above quite different. One of critical problem to address in Monitoring and Management (M&M) is to provide scalable monitoring. This means irrespective of size of Distributed system end-user should be able to effectively monitor the system.

In a typical monitoring scenario this will involve requests to set of mbeans and this set will repeat. So one possible optimization is that prefetch this information in one go. Thus number of remote requests will reduce from numberOfMbeans*NumberOfMemebers to numberOfMembers.

- **JMX Approach : JMX Proxies :** Here the granularity at which managing node operates is a Mbean proxy. So every request from JMX client is bascially federated to respective node. So typical scenario mentioned above will generate N request-responses. To implement above optimization a **Super Mbean** (one which operated at batch or coarse level) can be implemented.

  In built thread pool support.
- **Gemfire P2P Approach : Generic Request-Response Model :**This approach (current design) has very generic request-response model so above optimization can be easily implemented. Further other meta-data like “Mbean Set for Caching” can be easily accomodated. This will leave lot of hooks for further optimizations.

  To achieve thread pool support like JMX and execution separation from core Gemfire operations, a separate executor(DistributionMessage.processorType) has to be implmented in Gemfire.

- **Hybrid Approach :** This approach uses the same **Generic Request-Response Model** on top of RMI protocol. Each managed node exposes one Server object which is responsible for delegating the request to its handler.

  RMI specification for threading : A method dispatched by the RMI runtime to a remote object implementation (a server) may or may not execute in a separate thread. Calls originating from different clients Virtual Machines will execute in different threads. From the same client machine it is not guaranteed that each method will run in a separate thread"

Proof Of Concept

Above three apporaches were evaluated against each other with limited set of functionality implemented. POC involved a generic JMX client polling managing nodes for set of mbeans. Number of managed nodes were varied. While testing CPU utilization and response times were monitored. The mirror-mbean pattern implemented in POC (with some changes) is discussed here :
http://weblogs.java.net/blog/emcmanus/archive/2007/02/cascading_its_a.html

Results

In 50 node test response time was 6.5% worst for hybrid approach compared to JMX RMI connector response time while GP2P was marginally better (only 4% compared to JMX RMI). but CPU utilization was higher in GP2P approach. The Generic request-response building and parsing can be optimized further for reducing CPU usage.

Detailed results can be found in Appendix 2.

Appendix I

Gemfire P2P aproach has implemented flexible messaging model which is like Map Message or similar to JSON. This free-form Document like Object Model can help to implement features like Bulk-request, Any meta-data between nodes.

DynamicObject

```
public class DynamicObject  implements DataSerializable, Json ,
JavaSerializable{

      //Note method return same object so that method chaining is possible
      public DynamicObject put(String key, Object value);
      public DynamicObject append(String key, Object value);
      public DynamicObject copyProps(String[] key, DynamicObject
otherObject);

      //For transfer over networkstream
      public void fromData(DataInput di);
      public void toData(DataOutput dataOutput) ;

      //For interoperability with java beans
      public void fromJava(DataInput di);
      public void toJava(DataOutput dataOutput) ;

      //For interoperability with json. Helpful for debugging
      public DynamicObject fromJson(String str)
      public String toJson()

}
```

- Dynamic object mentioned above is actually nested HashMap.
- All inner objects are also Dynamic Objects. Non-Dynamic Objects can put using serialization spec(Not covered here, Not strictly required)
- Append operation will basically add another object against the same key/property, thus key/property is of type collection.
- Implements DataSerializable interface.
- Dynamic object can be directly from any other Java bean using serialization spec which is specified using JSON notation as follows

JSON Representation of the Bulk Reuqest

```
{
      type : "getMBeanData",
      mbeans : [
            {
                  objectName : "com.example:type=QueueSampler",
                  attributes : [ "cacheSize","diskQueueSize","readsPerSec" ]
            },
            {
                  objectName : "com.example:type=Hello",
                  attributes : [ "freeMemory" , "maxMemory" , "numThreads"]
            }
      ]
}
```

JSON Representation of the bulk response object

```
{
      type : "mbeanUpdatePush",
      mbeans : [
            {
                  objectName : "com.example:type=QueueSampler",
                  attributes : [
                        {attrName : "cacheSize", attrValue : 124} ,
                        {attrName : "numUpdates", attrValue : 123523}
                  ]
            },
            {
                  objectName : "com.example:type=Hello",
                  attributes : [
                        {attrName : "freeMemory", attrValue : 50} ,
                        {attrName : "maxMemory", attrValue : 64}
                  ]
            }
      ]
}
```

Appendix II

Stat NO	Number of Remote Managed Nodes	Request Number	JMX	P2P	RMI(Hybrid Aproach)	Number Of Mbeans queried	Total Attributes Queried	Total Number Of Requests
						1180800	4723200	2400
1	50	0	3061	5543	3494			
		100	1017	953	1078			
		200	890	862	984			
		300	806	782	880			
		400	755	683	774			
		500	752	726	756			
		600	729	722	809			
		700	730	747	758			
		800	732	717	804			
		900	742	714	761			
		1000	726	676	798			
		1100	726	713	792			
		1200	735	691	801			
		1300	718	707	798			
		1400	723	719	793			
		1500	727	707	765			
		1600	759	706	787			
		1700	725	709	793			
		1800	732	650	785			
		1900	723	716	794			
		2000	736	647	760			
		2100	722	698	791			
		2200	723	710	806			
		2300	725	654	752			
Total Time =			1852969	1774270	1975106			
Average =			754.47826087	722.13043478261	809.521739130			
CPU Us			26.00%	40.00%	30.00%			

Test Strategy : In each request all the Mbeans of the Managing nodes(including proxies) are queried for their attributes.
2400 such queries were carried out.

Every 100th request recorded. Hence 24 results shown above.
Each query is for all the attributes of all the Mbeans

An Mbean can be an JMX proxy of P2P proxy depending on the config
Each Mbean is a StandardMbe