Replication and load balancing strategy of STAR's Relational Database Management System (RDBM)

Michael DePhillips, Jerome Lauret

Brookhaven National Laboratory, Upton NY 11973

Mikhail Kopytine

Kent State University, Kent Ohio 44242

jlauret@bnl.gov

Abstract. Database demand resulting from offline analysis and production of data at the STAR experiment at Brookhaven National Laboratory's Relativistic Heavy-Ion Collider has steadily increased over the last six years of data taking activities. With each year, STAR more than doubles the number of events recorded with an anticipation of reaching a billion event capabilities as early as next year. The challenges faced from producing and analyzing this magnitude of events in parallel have raised issues with regard to the distribution of calibrations and geometry data, via databases, to STAR's growing global collaboration. Rapid distribution, availability, ensured synchronization and load balancing have become paramount considerations. Both conventional technology and novel approaches are used in parallel to realize these goals. This paper discusses how STAR uses load balancing to optimize database usage. It discusses distribution methods via MySQL master slave replication; the synchronization issues that arise from this type of distribution and solutions, mostly home-grown, put forth to overcome these issues. A novel approach toward load balancing between slave nodes that assists in maintaining a high availability rate for a veracious community is discussed in detail. This load balancing addresses both, pools of nodes internal to a given location, as well as balancing the load for remote users between different available locations. Challenges, trade-offs, rationale for decisions and paths forward will be discussed in all cases, presenting a solid production environment with a vision for scalable growth.

1. Introduction

The Relativistic Heavy Ion Collider (RHIC) STAR experiment [1] at Brookhaven National Laboratory (BNL) [2] has completed seven successful years of data taking. Much of these data are reconstructed and available for analysis. Both of these operations are dependent on calibrations and geometry data stored in MySQL [3] Relational Database Management System. With each passing year the load on these databases increased. STAR's original layout for these offline databases was a straight-forward design that included MySQL replication as a distribution mechanism and DNS Round Robin as a load sharing mechanism. As demand and load increased incremental steps were taken to ensure database availability and quality of service. Although useful, these actions only provided temporary relief. Developing load balancing software was viewed as a more scalable, and permanent solution.

This paper describes a common problem faced by users of distributed databases in a production environment, that being how to economically scale and how to maintain quality of service and service availability as usage increases. We describe a software component design, based on a policy driven system that provides centralized configuration and addresses these issues. This system proves to be a scalable solution by providing the means to optimize usage by adapting to the dynamic nature of STAR's environment.

This software provides a policy based distribution of database load that is centrally managed by an XML configuration file. It contains a rich set of features that an administrator can instantly adjust. The XML is interpreted by an updated version of STAR's C++ database API, which in turn makes a decision on which server to connect to. The cost of this operation is negligible, at most 20 milliseconds per job. This load balancing extents to a global community by including remote users with a local database and also provides an opportunity to include databases managed by remote users.

2. Background

The previous STAR database server configuration was conceived of, and implemented early in the life cycle of the experiment. By design, STAR used a Tier-0 model approach for its database servers. This meant maintaining a small pool of database servers centrally located at BNL. Redundancy was ensured via MySQL master/slave replication. When needed, stress relief for the servers was obtained by making minor hardware adjustments, query optimization and improvements to the STAR database API (API) [4]. This system provided a steady service of high quality and availability for six production years. Policies governing connections were minimal and only needed to be enforced through observation, monitoring and emailing requests to users.

As the life cycle of the experiment progressed, which included seven years of successful data taking, database-dependent operations increased. Raw data is reconstructed at an almost constant rate and the amount of data to analyze has increased proportionally. With this increase of usage and production the landscape of STAR's computing model began to evolve. Part of this evolution included the emergence of outside institutions (i.e., external to the Tier-0 center, BNL) providing their own computing resources. Thus the experiment began to take advantage of a distributed computing model. Database slaves also emerged at various institutions. At this point, however, a lack of controlled access to these smaller installations and the heterogeneity of the hardware across the landscape made it difficult to leverage these database resources into a global, distributed configuration.

As usage increased, availability decreased therefore the quality of service was beginning to be jeopardized. This was, apparently, not due to a lack of hardware, but due to the disproportionate use of resources. Constraints on an institutional level, and the inability to leverage global nodes caused some databases to be under used while at the same time other servers were over used and therefore stressed causing a potential bottleneck. Improvements to the system were made, however, since there were no global policies that governed all the database servers in the STAR network; they were done on an ad-hoc basis, usually in response to an emergent bottleneck.

2.1. Project Motivation

As the interval of these threats to the database service became shorter, it became apparent that STAR needed a clear plan to optimise the usage and maximize the through-put of its database servers. The first action was to establish a group of connection policies that had a flexible configuration which would support a sustainable system. Access to this configuration has to be limited to an administrator. These connection policies should hook into the API which would then load balance between available nodes. This load balancing should occur at a local institution level or in the global level for users outside of the local networks. Furthermore, all database nodes, regardless of their power or their locations, should be leveraged in the load balancing system. This would help recover and maximize use for all available resources, even on a global level.

The requirements that went into developing this set of policies are listed below:

- Flexible Groupings of servers

- Distributed computing paradigm
- Heterogeneous hardware environment
- Include different types of usage for databases

The policy of grouping nodes was incorporated into the original design. However, the definitions of those pools needed to be flexible. Leveraging the computing power outside of the BNL internal network would be advantageous to users from both a performance and consistency perspective. The hardware that makes up the nodes in these external networks is under local control, therefore, the final product must account for different grades of equipment (e.g., cabling, NIC cards, hard drives, etc.). Supporting a heterogeneous environment also allows an administrator to include more resources into the system under central control. Different data (e.g., administrative, calibrations, file catalogue) are stored in different databases and therefore the final product must be able to support the diversity of these different systems.

2.2. Original Configuration

The original approach was to use MySQL master slave replication and push the STAR calibration and geometry databases to as many nodes as needed to provide adequate service [5]. Scalability was achieved by adding nodes on an as-needed basis. Over time this worked out to include one master, used for writes only, one development machine used by a limited number of people and eight slaves which serves approximately 1000 processors used by approximately 50 collaborators.

Of paramount importance in both the original and any subsequent configuration is the ability to produce analysis data files ("PROD") from the original raw data files by the fastest and most stable means possible. This implies that there must be a separation between the defined times of PROD and the random nature of general users analysing ("ANA") these files. Both PROD and ANA rely on database services, therefore a decision was made to divide the available slaves into two distinct and isolated pools, one for PROD the other for ANA

The policy governing the destination of which job to which pool was determined by a small configuration file. This file, constructed in XML format, could be found by the database API in the following order of precedence; in a user-set environmental variable; a user's home directory; or at a default location in the STAR environment. There was no way of ensuring that users adhere to the policy and make the correct choice in their personal configuration file or environmental variable.

A load-sharing scheme using DNS-Round Robin was implemented within each pool. The results of this sharing can be seen in figure 1 which show the number of MySQL connections as a function of time, for two nodes in the ANA pool. As expected, load is shared with approximate accuracy; however, figure 2 clearly demonstrates the problem of under-used resources on a node assigned to the separate PROD pool. This plot was made at the same time as figure 1. Clearly, the ANA pool is being stressed; note the red color emphasizing the effect of the connections. Meanwhile the nodes in the PROD pool sit relatively idle, which is displayed by a lower number of connections accented by the more benign color scheme.

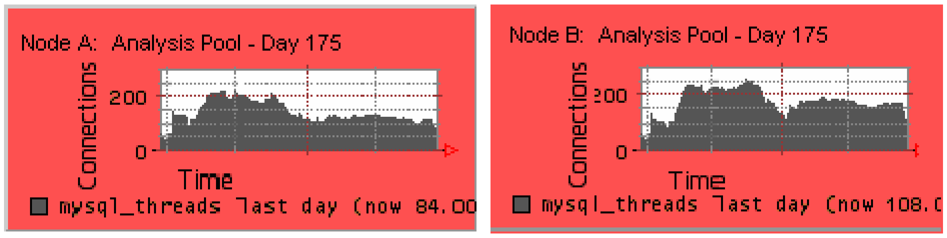

Figure 1. Existing connections versus time for the DNS round robin analysis pool

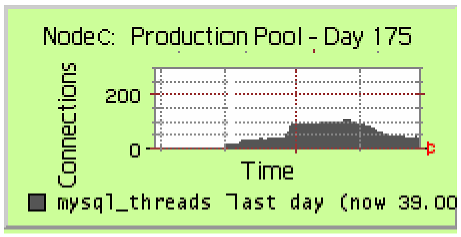

Figure 2. Existing connections versus time for the DNS round robin production pool

2.3. Early Relief Strategies

Since DNS round robin was controlled on an institutional level, it was impossible for a STAR member to move nodes into and out of pools on an as needed basis. Thus we had to live with periodic disproportionate usage and look to alleviate stress in other, less obvious, ways. In particular, ad-hoc optimization techniques where employed when the results of this stress became noticeable to users. These techniques included query optimization, adding memory or disks, server parameter tweaking, and reviewing and recoding parts of the API. All of these were worthwhile efforts serving the immediate purpose of relieving stress while providing the longer term benefits of improving the database infrastructure. They did not, however, address the issue of underused servers. Also ignored were database servers and users outside of the institution. Most importantly there was still no policies governing these servers. All actions, maintenance and upgrades, were reactive addressing problems and bottlenecks as they arose. Options for quick response and relief began to dissipate over time.

As opportunities for optimization and code improvements became exhausted, we were forced to consider adding more slaves. This was not a welcome option due to the cost and limited scaling potential. It was also difficult to justify additional hardware when half of the existing nodes were sitting idle at times. Looking for solutions outside of the major institutions, we encouraged smaller institutions to maintain and use a slave local to their institution. This removed some users from a saturated system by adding additional database slaves to the STAR community. It soon became evident that these new slaves were only accessed by a few users thereby the servers were undoubtedly under used. Again, the fundamental motivations were not being addressed so in order to incorporate the points mentioned in section 2.1 a load balancing software project was undertaken.

3. Load Balancing Software

The software was designed to load balance between all available nodes. We wanted to divide available nodes into clearly defined pools with a rich set of attributes that map to programmed features. These features should be able to be adjusted and the pools should be able to be quickly altered, via a centrally managed configuration file. This configuration file should be a verifiable and validated XML file. This file should be only edited by administrators monitoring the system; therefore, set policies cannot be easily circumvented. Different types of databases (e.g., administrative, file catalogue, etc.) should be able to take advantage of the software. Also, STAR users that do not log into a major institution and connect to a database, via an open port into these institutions, should be able to load balance between all globally accessible databases. Individual database slaves, being maintained by smaller institutions, should be included in this global load balancing.

3.1. Programmed Flexibility / Features

Access to STAR databases varies greatly with regard to time, users, and types of request. It also varies with regard to the types of machines being accessed and their location with regard to network. As mentioned in section 2.1, leveraging this diversity was a project requirement. The use of a configuration file using validated XML provides the load balancer with a direct mapping to the overlying strategies of the software architecture. The concept of a distributed architecture was achieved by the naming of the XML file and will be discussed in detail in section 3.2. The remaining requirements can be achieved within the actual XML using the following features, elements and attributes:

- Pool creation – via DNS name and port number
- User identity – restrict a pool of databases to a list of users
- Access mode - restrict a pool of databases based on their intended usage (i.e., read/write)
- Usage type - restrict a pool of databases based on type of user/jobs (e.g., PROD)
- Time of day - activate a pool of databases with a granularity of day or night
- Day of week - activate a pool of databases on a particular day(s)
- Connection limits – set a cap on allowed users on a pool of databases or individual node
- Weighting factor – adjust the proportion of accepted connection of a particular node with respect to other eligible nodes

We adapted the concept of pools of servers from our original configuration; however, by adding attributes and the ability to dynamically move nodes in and out of these pools we eliminated the limitations of DNS round robin. The pools are defined by the element *Server* which can contain one or more *host* elements. These elements have attributes associated within that scope of the grouping. For example, individual nodes are identified by a *name* and a *port*. They can also have a *cap* or *machinePower* assigned to it. As explained in further detail below, these attributes enable an administrator to group together different grades of hardware, thus satisfying the requirement for allowing a heterogeneous environment. The server level has attributes such as *scope*, *whenActive* and *user*. These help to determine who has access to the pools and when. This architecture can be viewed in the schema displayed in figure 3.

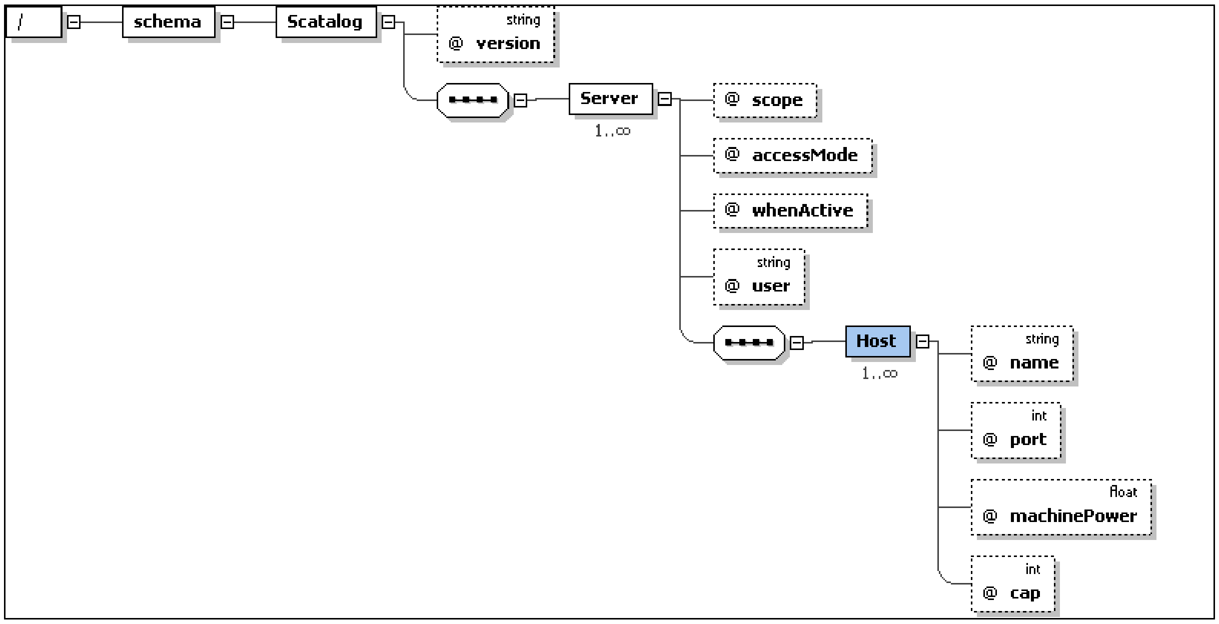

Figure 3: XML Schema

Figure 4 displays a portion of a typical XML configuration file that sets these elements and attributes.

```
<Server scope="Production" user="recons,john,paul" accessMode="read">
                <Host name="2db.bnl.gov"  port="3333"/>
                <Host name="3db.bnl.gov"  port="3333"/>
                <Host name="4db.bnl.gov"  port="3333"/>
                <Host name="5db.bnl.gov"  port="3333"/>
           </Server>

           <Server scope="Analysis" accessMode="read">
                <Host name="7db.bnl.gov"  port="3333"/>
                <Host name="6db.bnl.gov"  port="3333"/>
                <Host name="8db.bnl.gov"  port="3333"/>
           </Server>
<!--
           <Server scope="Analysis" whenActive="day" accessMode="read">
                <Host name="1db.bnl.gov" port="3333"/>
           </Server>
-->
          <Server scope="Analysis" user="john,paul,george,ringo" accessMode="read">
                <Host name="1db.bnl.gov" port="3333"/>
           </Server>
```

Figure 4: A sample XML configuration

We added to our C++ library the standard library libxml2 which gave our API the ability to easily parse this XML file. The API then connects to each database server in the appropriate group, and executes the MySQL command *show processlist*. This command reports the number of connections. The API then counts and compares that number with the other available servers, picks the lowest and makes a connection for the user. Certain attributes such as a weighting factor called *machinePower*, discussed in detail below, will alter the outcome of this algorithm. The time it takes to execute these operations measure on average of 0.01 seconds with a maximum measurement of 0.02 seconds. The results are displayed in figure 5 which shows a Ganglia [6] plot of the number of MySQL connections as a function of time.

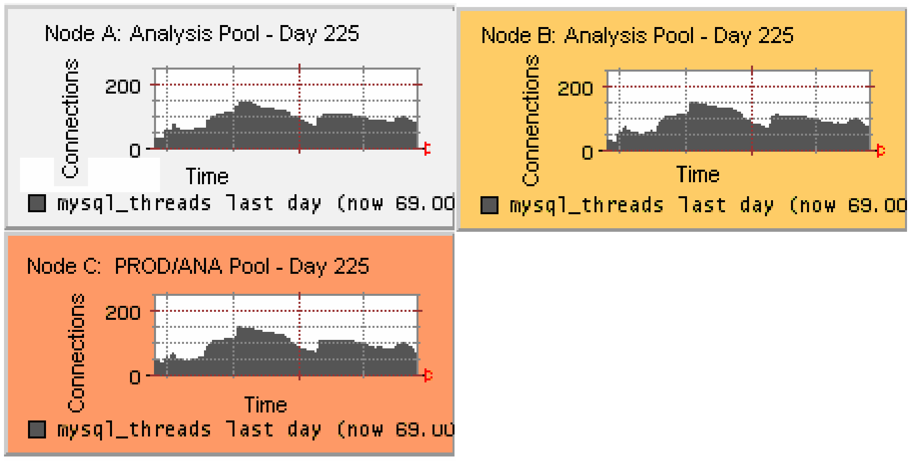

Figure 5. Load balanced Severs show similar load as a function of time

An example of the practical functionality of these attributes could be seen in Figure 6, which shows the result of setting the *whenActive* attribute equal to “Night” for the development pool which at the time contained one node. Specifically this forces our development node to remain isolated to developers, listed by name with the *user* attribute, during the work day, but allows the node to be accessed by general users between 23:00 and 07:00.

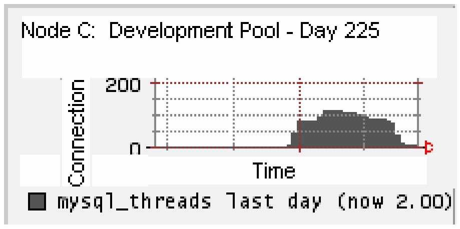

Figure 6. Server using the attribute and value *whenActive=Night*

Not displayed in figure 4, but of particular interest is the attribute *machinePower*. The benefits of this weighting mechanism are twofold. First, by increasing the machine power of a node, connections are proportionally attracted to it. That is, if a job or a series of jobs, from a specific group of users, exhibits unusual load pattern, and a flooding of all the servers is detected, an administrator can create an ad-hoc pool for those users, allocate the node(s) and assign a high *machinePower*. This will direct future jobs of those users into this quarantined pool. The user(s) responsible for this flooding will suffer delays but their jobs do not have to be killed. A similar effect could be achieved by assigning a single node to that user(s). Second, in the distributed computing paradigm now used by STAR it is impossible to enforce a consistency of hardware across the computing landscape. Therefore, in order to effectively load balance the software must compensate for different grades of equipment. If a node is older than and not as robust as the other nodes in a pool, the machine power could be reduced. This would reduce the proportion of requests the server receives therefore, providing a fairer load share. A similar effect could be achieved by regulating the connections with the *cap* attribute.

3.2. Local and Global Configuration

As mentioned, the load balancing and resource availability is centrally controlled via an XML file. In fact, there are two types of XML files; one for local configuration and one for global configuration. This separation allows for multiple instances of the local file at various sites. For a local configuration, that is within an institution that has more than one database server, this file is placed on a shared disk and pointed to by an environmental variable. This configuration will be available to all users logging into that institution and provide a configuration tailored to that institution.

There are also instances of users who have personal copies of the STAR environment and libraries which need to be considered in a distributed design of the database configuration. Data access for these users comes from one of two ways: 1) Users can maintain a slave of the STAR databases; 2) users can access the data through a conduit in a major institution's firewall. This brings up two separate issues when considering a distributed paradigm, how to include the slaves from the smaller institution into the systems and how to provide load balancing to users with their own environment (i.e., not logged into an institution with a local load balancing configuration).

If no local configuration is present, global load balancing is activated with a single instance of a global configuration file, also pointed to by an environmental variable. This file is available to users outside of the internal STAR network via AFS or via HTTP if AFS is not installed on the remote site. This file is identical in structure to the local configuration file, except that it lists all available open ports to various institutions. A global user now load balances between the various databases located throughout the world. The precedence the API uses for checking files is as follows, it first checks for a local configuration, if there is none it checks for the global configuration if neither are present it falls back to the older method of connecting.

Smaller institutions that chose to maintain an individual slave have no need to load balance or set policies for the individual node, however, this node will most likely be idle some percentage of time. The load balancing policies provide enough assurances for these institutions to offer their node for global use. For example, the connection *cap* attribute will ensure the machine won't get flooded, the dynamic nature of the configuration file will allow a host to be removed from the list of available nodes and the time attributes define when a node is available.

4. Path Forward

This is the initial release of the load balancer to the API, further development is already underway.

4.1. Decision algorithm

Revisiting figure 4, we note that although the connections are balanced, the colour of each plot shows that there are different loads on each of the nodes. Obviously, to increase the fairness of the load balancing it would be beneficial to include machine based statistics (e.g., I/O, CPU usage, total load etc) into the decision making process.

4.2. Abstraction

Abstracting the code that parses the XML and decides upon a connection into its own library will reap the typical benefits of modularization. Also, making the code independent of a specific database application would allow the code to be used by other types of services.

4.3. Monitoring

The core of the decision making algorithm is based on real time dynamic information retrieved from each node in a system. These data are also useful for monitoring, database and machine optimization. Expanding existing monitoring tools to include these data could further enhance the system.

5. Conclusions

Over the course of seven successful data taking runs, the initial STAR calibrations and geometry database server configuration began to reach its limitations. This original configuration was replaced with a policy based, centrally managed, load balanced system. This system provides a scalable path forward that will ensure availability, through maximizing all existing resources and maintaining a framework that will easily absorb additional nodes if needed. This functionality extends beyond the institution and incorporates a global arena, where a user can take advantage of balanced global resources. Furthermore, all STAR database resources that are distributed throughout the world can be included in this accounting.

The measurements of the effectiveness of the project show it to be working. Stress has been relieved on the STAR databases and the overall overhead of the application is minimal. Paths for improvement and abstraction seem promising and imminent.

References

[1] The Solenoid Tracker At Rhic (STAR) experiment – http://www.star.bnl.gov/

[2] Brookhaven National Laboratory is operated and managed for DOE's Office of Science – http://www.bnl.gov/

[3] MySQL http://www.mysql.com

[4] STAR database documentation http://www.star.bnl.gov/STAR/comp/db/

[5] **DePhillips M, Lauret J. Perevoztchikov V. Porter J.** *Proc. 2006 XV International Conference on Computing in High Energy and Nuclear Physics (CHEP-06)* **vol 1 (Macmillian India Ltd.)** p 80

[6] Ganglia Monitoring System http://ganglia.sourceforge.net/

[7] **Porter R. J.** *Proc. Of CHEP 2001 Beijing* **(CERN 2001-2-030)**