Load Balancing for Hypertable

Gordon Rios
Cork Constraint Computation Centre
University College Cork
Cork, Ireland

Doug Judd
Hypertable, Inc.
702 Marshall St.
Redwood City, CA

Abstract

In Hypertable ranges of table data are stored and accessed on different nodes and allows for flexible management of the underlying hardware. Overall performance is sensitive to the balance of range load across the cluster. The project developers aim to create a simple interface to allow researchers to design experimental load balancing strategies that incorporate machine learning and optimization. This paper specifies the load balancing problem and introduces it as a challenge problem for AI and machine learning.

Introduction

Hypertable is an open source, distributed, column oriented database, designed to run on clusters of tens, hundreds, or thousands of nodes (Judd 2007). The project is closely modeled after Bigtable, the proprietary database system developed by Google. Bigtable has been a highly visible component of Google’s success as a company (Chang et al. 2006) and has increased the popularity of distributed datastores. The essential operation of distributed datastores is writing new data to a cluster of servers managing redundant partitions of the table space supported by the system. The developers of Hypertable have created a flexible interface allowing for customized implementations of load balancing for Hypertable.

Load Balancing for Hypertable

The current range allocation algorithm is round robin. Load balancing is performed after new range servers are added, when a range splits, and periodically according to cluster policy. The objective of load balancing for range servers in Hypertable is to ensure that operational resources, as measured by CPU load average (loadavg), are balanced across the cluster. CPU load average is reported over 1-, 5-, and 15-minute intervals (Walker 2006) and has been investigated as a metric for evaluating dynamic load balancing (Ferrari and Zhou 1987). It is an exponential moving average of the number of processes that are running (or runnable) per CPU.

Basic Balancer

As an illustration we introduce the basic load balancing algorithm (basic balancer) that serves as a baseline reference implementation. The algorithm is described on the project wiki3 and is designed to reduce the dispersion of loadavg across the set of range servers. The basic balancer cycles through each server in descending order of loadavg and moves ranges from the highest to the lowest as long as that move reduces that server’s estimated deviation from the mean loadavg for the cluster. It is estimated since the precise effects of removing the range from one server and adding to another are uncertain.

Estimating a Range’s Effect on Load Average

Clearly, an algorithm for balancing loadavg needs to estimate the effect of a specific range on a given range server in terms of loadavg. For the basic balancer a simple loading factor computed per server is used. For each range a heuristic estimate (loadestimate) of its contribution to loadavg is computed as:

\[ 2 \cdot \text{bytes\_written/s} + \text{disk\_bytes\_read/s} \]

where bytes\_written/s is the raw number of bytes that get written to a range per second. It is multiplied by two because there are at least two writes for each piece of data written to the database.4

For read activity on the range disk\_bytes\_read/s is used which is a measure of uncompressed bytes transferred from disk during query execution. It captures load effects of different compression schemes configurable for each table. This is done to normalize the number of bytes read in relation to bytes written since the bytes get compressed prior to being written to disk.

Next, a loading factor for each server \( i \) (loadavg\_per\_loadestimate) is calculated as:

\[
\text{loadavg}_i = \frac{\sum_{j=1}^{n_i} \text{loadestimate}_{i,j}}{n_i}
\]

References

1http://code.google.com/p/hypertable/wiki/LoadBalancing
2http://linuxtechsupport.blogspot.com/2008/10/what-exactly-is-load-average.html
3http://code.google.com/p/hypertable/wiki/LoadBalancing
4First a write for the commit log and then one for a CellStore. Further, there is the possibility of additional load due to compactions and ambient load due to distributed file system writes for copies. For full details see (Judd 2011).
where \( n_i \) is the number of ranges at server \( i \) and \( \text{loadestimate}_{i,j} \) is the loadestimate of range \( j \) on server \( i \). Thus, the approximate change to \( \text{loadavg} \) is determined by multiplying a range's loadestimate by the assigned server's \( \text{loadavg} \).

**Machine Learned Estimates for a Range's Effect** We expect that more accurate estimates, using specific properties of the given range and range server combination would improve balancer performance. One challenge for research would be to construct machine learned models for estimating the effects of adding or removing a specified range from a particular range server based on properties of the range and recent operational statistics of the range server. A comprehensive survey of machine learning applications to cluster problems such as estimating system work loads can be found in (Bodik 2010).

Statistics usable as features in a machine learned models or other balancer inputs or are stored in the sys/RS_METRICS table. Currently, Hypertable collects the following application level stats both on a per range and per table:

- range count
- scans or updates per second
- bytes read per second
- disk bytes read per second (\( \text{disk} \_\text{bytes读 per second} \))
- bytes written per second (\( \text{bytes written per second} \))
- disk used
- memory used

For the range servers, statistics including CPU load average, are collected via Hyperic SIGAR\(^3\) and used as inputs and feedback for load balancing operations. For example, SIGAR reports approximately 50 different measurements (e.g. load average, CPU, memory, network utilization, disk inputs/outputs per second (IOPS), bytes read/written per second, paging statistics, etc.). Stats collected for each range server include, but are not limited to: syncs per second, block cache hit\%, query cache hit\%. In addition, general server properties are known including amount of RAM, number of CPU cores, and disk capacity.

**Optimizing the Load Balancing Plan**

Load balancing has been studied from a variety of standpoints (Tantawi and Towsley 1985; Mehra 1992; Parent, Verbeek, and Lemeire 2002; Parent et al. 2004; Keslasy et al. 2005; McDonald and Turner 2000). Initially, it is assumed that optimization will be batch oriented producing a balance plan as output. The balance plan is an assignment of subsets of ranges to be moved to a subset of the available range servers.

---

\(^{3}\)http://www.hyperic.com/products/sigar.html

**Objective Function for Load Balancing** A variety of objective functions and online learning methodologies have been described for load balancing. An exhaustive survey is well outside the scope of this paper but a small sampling of ideas are discussed in (Mehra and Wah 1993; Westbrook 1995). For Hypertable, one simple objective for the load balancer, and one consistent with the algorithm outlined for the basic balancer, is the sum of absolute deviations of the \( \text{loadavg} \):

\[
\text{Deviation}(\text{loadavg}) = \sum_{i=1}^{N} |\text{loadavg}_i - \overline{\text{loadavg}}|
\]

for \( N \) range servers where \( \overline{\text{loadavg}} \) is the sample mean across the cluster. Alternatively, we can consider the range defined as:

\[
\text{Range}(\text{loadavg}) = \max(\text{loadavg}) - \min(\text{loadavg})
\]

**Constraints for Load Balancing** Constraints for allowable range moves should be considered. Rules for restricting range assignments based on properties of either range or range servers should be supported. Costs for range moves could be estimated and factored into the assignments as well as constraints on the total number of moves that can be made.

**Test Load and Experiments** Experimental test loads with a variety of profiles is under development and is located on the project web site.\(^6\) Some of the load is drawn from real time streaming services such as twitter. Twitter stream data is being actively researched for a variety of purposes related to community mining (Wu et al. 2011) and scalability challenges (Eriksen 2010) of the sort addressed by Hypertable.

**Acknowledgments**

The authors wish to acknowledge the helpful comments of Sanjit Jhala\(^7\), Charles Sutton\(^8\); and Luke Lu\(^9\) for early discussions on the intelligent load balancing API. Gordon Rios is sponsored at the Cork Constraint Computation Centre by a gift grant from Gino b.v. in Groningen, NL.

**References**


\(^{6}\)http://hypertable.org

\(^{7}\)Hypertable, Inc.

\(^{8}\)University of Edinburgh

\(^{9}\)Yahoo!, Inc.


