

# Load Balancing on a Grid Using Data Characteristics

Jonathan White and Dale R. Thompson  
Computer Science and Computer Engineering Department  
University of Arkansas  
Fayetteville, AR 72701, USA  
{jlw09, drt}@uark.edu

## Abstract

*In this paper, we develop an efficient partitioning scheme for a grid environment to increase performance by measuring the characteristics of the data. We design a model that simulates a real life distributed grid environment, and test this model against a synthetic data set. We use public information about the distribution of U.S. zip codes and last names to make an effective partitioning scheme for a database running on a grid environment. We demonstrate that smaller data partitions are more effective at distributing loads evenly across the grid resulting in a quicker response time.*

## Keywords

Load balancing, partitioning, grid technologies, distributed databases

## 1. INTRODUCTION

Grid technologies are becoming commonplace [1]. The typical grids are computational grids that provide large computational resources at a fraction of the cost of large servers. Another use of grids is memory-based databases that store records in memory instead of disk to increase performance. With the growth of grid technologies, more and more companies are moving from large scale, centralized databases to databases that reside on grid-based systems.

One of the most important benefits that a grid can provide to the users of a database is the ability to process requests with a high degree of parallelism to minimize the mean response time [2]. One of the challenges that these companies face when migrating their database from a centralized environments to a grid environment is how to partition their data across the computers in the grid to promote load balancing and parallel retrieval.

The first challenge of distributing data across a grid of computers is load balancing. Load balancing strategies try to distribute the workload uniformly across all computers in a grid [3]. Load balancing can be done without measuring the current load to avoid the overhead and temporary balancing as in [4]. Many load-balancing

strategies dynamically react to load imbalances by comparing a load metric to a threshold and transferring workloads to other computers if the threshold is exceeded [5]. Other load balancing strategies use workload priorities and characteristics of the workload to do load balancing [3], [6]. Collecting workload characteristics in advance can decrease the mean response time of batches of requests [6].

The second challenge of distributing data across a grid is how to promote parallel retrieval from different computers in the grid. A common approach is related to the strategy called declustering, which partitions the data and allocates it to multiple disks [7], [8]. The data is intelligently partitioned and assigned to the computers in the grid to promote parallel processing.

Typically, companies have a wealth of information about the characteristics of their data. This knowledge is useful for customer profiling, fraud detection, and evaluation of retail promotions. In addition, this information can be used to partition the data across a grid to improve performance.

The paper is organized as follows. In Section 2, we present a motivating scenario to demonstrate where partitioning based on characteristics of the data would be appropriate. Then in Section 3, we formulate a potential solution to the motivating scenario. In Section 4, we describe how we implemented the solution. In Section 5, we describe the results that were obtained for the motivating scenario. Finally, in Section 6 we present our conclusions and point out future research areas.

## 2. MOTIVATING SCENARIO

Consider a company with a very large customer database that mails offers on a regular basis. When responses are received, the company must match the name and address given on the response to those in the database in order to identify the respondent and to update his or her information. If the company receives several thousand responses to its offer each day, a traditional relational database management system might not be able to process the responses fast enough, and consequently, the company might lose business to its competitors.

The current database resides on a large centralized database system, but the company would like to move it to a grid architecture. They receive a large amount of batch requests that contain multiple records to be verified. Mean response time of the batch requests is a major concern as they need to process the applications as soon as possible or they will lose business to their competitors.

Assume the company owns 100 identical node computers that are connected in a grid architecture. They need to partition their database across the grid in such a way that the mean response time of the system is minimized. The application can handle a block of approximately 2,000 records at a time and must wait until all records in the block are processed before it can process another block. In addition, they must process and return the requests in the exact order that they were received.

With these given parameters, we will demonstrate some effective ways that the company can partition their database across a grid to minimize the time to process a large batch of records.

### 3. SOLUTION

The proposed solution partitions the database across the grid evenly according to the distribution of U.S. zip codes and last names by analyzing publicly available data to partition the data. We compare two different methods. The first method uses distribution information based on zip codes to distribute the database. The second method uses information based on the distributions of U.S. zip codes *and* last names to distribute the database more uniformly.

We demonstrate that the overall response time to process a batch of records is significantly decreased using the information available about the distribution of U.S. records. Partitioning using both the zip code and last name distributions processes a batch of records faster than partitioning based only on zip code distributions.

This is the proposed solution. For a system that uses information about just U.S. zip codes, we attempt to distribute all the records based upon their zip code evenly. We want every computer in the grid to have approximately the same number of records.

We also want to encourage parallel processing and balance the processing across all grid computers. To do this, we use the zip code modulo two times the number of nodes to form a partition. Then, we match the largest partitions with the smallest partitions to maintain balance across the grid.

This partitioning scheme helps to ensure that numerically close zip codes are *not* on the same node. For example, all the zip codes from Illinois would tend to be distributed across the grid uniformly. This helps load balance batches of records that are sorted by zip code or name. An example a zip code base table is shown in Table 1.

**Table 1: Example of the zip code base table.**

<u>Zip Code</u>	<u>Node Address</u>
72701	27
72702	28
72703	29
72704	30

The partitioning scheme that uses information about both the U.S. zip codes and the last names is a slight modification of the above method. The first scheme only needed the zip code to partition; this method needs the zip code and the last name of the record. We will still use the idea of taking the zip code modulo two times the number of nodes in the grid and pairing up the largest partitions with the smallest partitions on a node. The result is stored in a table for directing requests and is referred to as the “zip base” table. This table directs a record to a particular grid node for processing based upon the zip code.

The distribution of U.S. last names balances the load by adding a name offset table. The name offset table separates all the U.S. last names into equal range partitions based on the number of grid nodes. For example, if there are 100 nodes, 1 percent of the last names will be in each range as seen in Table 2.

**Table 2: Example of the first seven entries in the name offset table.**

<u>Offset</u>	<u>Name Ranges</u>
1	AAAAAAAAA ALI
2	ALICEA ANDERSON
3	ANDERTON AVERETT
4	AVERILL BARBER
5	BARBERA BEACH
6	BEACHAM BENTON
7	BENTZ BLANCO

One percent of the U.S. population lies in the first line, 1 percent lies in the second line, and so on. The number to the left of the line is the offset. The offset indicates how far from the base zip an individual record should be. For example, if the record were “Jon Anderson”, with zip code 72701, the system would look up 72701 in the zip code base table to find the base address of 27. Then, the system would then find the offset for “Anderson” from the offset names table and get two. Adding 2 to 27 modulo 100 equals 29, and this is where this record would be located on the grid.

Compared to only using zip codes to partition data, the new scheme requires modifying the initial loading of the data onto the grid. Instead of loading a zip code modulo two times the number of nodes onto a node, you would need to

load smaller partitions based on the range of names. The initial loading routine would need access to the zip code base table and the name offset table to do this. If there are 100 grid computers, the basic unit of division is 1 percent of a given zip code.

In addition, the method for routing requests to grid computers would be modified. The routing routine will now need to do two table lookups and then add what it finds in the tables to calculate the final address. When the name offset and the zip code base are added together, it will be necessary to add modulo two times the number nodes. For example, if a zip base was 90 and the name offset was 20 and there were 100 nodes, the system would do the operation  $(90+20) \pmod{100} = 10$ .

The goal was to determine a partitioning scheme that provided better load balancing than using just the distribution of U.S. zip codes and decreased the mean response time with minimal additional overhead. The proposed scheme requires two table lookups and one addition to determine the correct computer in the grid for directing requests. This partitioning scheme is very fast and requires a minimal amount of memory.

## 4. SOLUTION IMPLEMENTATION

In this section, we discuss the details of our implementation to the problem, given the constraints in the motivating scenario. We describe how to make the zip code and names table. Then we describe how we modeled the two schemes. Finally, we describe how we tested the two schemes against various client input files.

### 4.1 Design of Zip code and Last Names Tables

We first had to find information about the distribution of U.S. zip codes. This was required to model a system that partitions the database across a grid based on zip codes. We needed to know how many U.S. zip codes there are, and how many people live in each zip code. This information is available from the U.S. Census Bureau's website [9]. The file that we used listed the population for every zip code in the U.S., along with other information including zip code.

We then wrote a program to distribute a database of records by finding the zip code modulo two times the number of nodes, matching the largest partition with the smallest partition, and putting the records on a grid computer. The populations indicated in the Census Bureau file determined the largest and smallest partitions. The result of this program was the zip code table that is used to direct requests to the proper node. There were approximately 29,000 zip codes.

We also found information about the distribution of U.S. last names from the U.S. Census Bureau web site [9]. The

U.S. Census Bureau surveyed 7.2 million Americans, and found approximately 88,000 unique last names. Smith was the most common last name, representing 1.1 percent of the U.S. population.

Then we sorted the last names alphabetically and formed offset ranges based on the number of nodes. For example if there were 50 nodes, we wanted  $1/50 = 2$  percent of the U.S. population to be represented in each range so that we could spread each zip code across the grid uniformly. Beginning at the start of the sorted file, we kept adding the name percentages until we reached 2 percent. The name that we started with and the name that we ended with formed the range. For example, the first range was AAAAAAA-ALI, because all the U.S. last names in this range summed up to be 2 percent of the U.S. population. The result was a names table that would be used to direct requests. If the number of grid computers changed, the names table could be dynamically reconfigured given the new number of computers.

### 4.2 Models to Test Solutions and Collect Data

We simulated the two partitioning schemes to determine the response time over workloads with varying distributions of records. Until all records have been processed, the client submits blocks of 2,000 records from larger batch of records. Based on the zip code or zip code and last name of the record, the blocks are directed to the appropriate grid computer to be processed. It is assumed that each record requires the same fixed processing time. We measure the simulated total response time, which is the time it takes for all records to be processed. The more records that the system can process in parallel, the lower the response time will be. We compare the response times of the two partitioning schemes to see which one is faster. We assume that every client request can be found somewhere on the grid, that is, that our information is complete and 100 percent correct.

Given the constraint that the system must wait until all records in the block of 2,000 are finished before it can process another block, the response time for an individual block is determined by the node that must process the largest number of records at that particular point in time. For example, if one node processes 1,500 records in the block of 2,000 and the other 500 are on other nodes, then the response time for that block is 1,500, because that is the total amount of record processing time that the system must wait for all the records in the block to be processed.

### 4.3 Design of Client Input Files

We decided to use client input files that came from the general U.S. population to simulate a workload. We also wanted the client file to be similar in size to the ones that a

real system might process. We decided to make the test client files 2 million records in length.

The test files needed to be realistic by following the distributions of records that contain U.S. zip codes and names. The test files had to fit the U.S. population distributions. If .01 percent of the U.S. population lived in a Chicago zip code, .01 percent of the test file should come from that zip code if the test file was to represent a realistic workload. The same data from the U.S. Census Bureau used to make the zip code table was used to find the distribution of zip codes and names.

To make an approximate representation of a typical workload containing U.S. records, we multiplied the percentage of people that lived in a particular zip code times the number of records in our test client files (2 million). This gave us the number of people, given that you were sampling from the entire U.S., in a file of 2 million records. We did the same operation for the name distribution as well.

We randomly paired a first name, last name, and zip code. For any sampling of the file, the distribution approximates the distribution of zip codes and names in the U.S.

The records in the client input files were sorted in three ways: randomly, sorted by only zip, and then sorted by zip and then last name. The real life system would have to face workflows of all the above scenarios. Workloads are commonly sorted and can have a significant impact the mean response time.

## 5. RESULTS

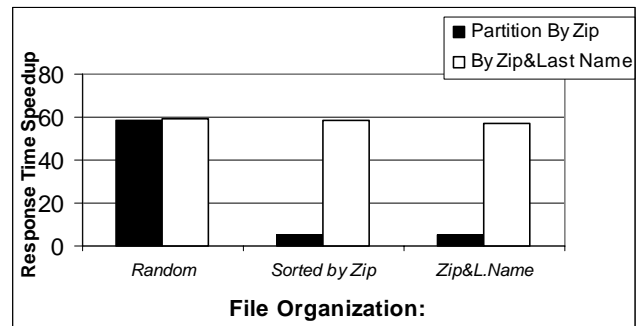
The results demonstrate that, while using zip codes to partition the data was effective when the system processed client input files that were not sorted, the method of partitioning by zip code *and* last name was better than the method of partitioning by zip code alone. The method of partitioning by zip code and then last name had a lower response time than the method of partitioning by zip code only. The response time for the method of partitioning by name and zip code is lower because partitioning by zip code and last name more evenly distributes the requested records. This makes the system more capable of handling sorted files, and the system is still able to handle random files with the same response time as the method of partitioning by just zip code.

The results also demonstrate that using the characteristics of the data can be used to make efficient, partitioning schemes that can be applied to a grid environment to increase performance. The routing of the requests is fast, takes a minimal amount of resources to implement, and is scalable. Using data about the workload can be used to increase performance.

## 5.1 Response Time for Client Files

In this section, we compare the response time of the two different schemes of partitioning. We examine how the system responds to a client file that has a distribution that matches the entire United States. The result is shown in Fig. 1. In addition, the two partitioning schemes are compared on the same client file sorted three different ways: random, by zip code, and by zip code and last name. The client input file contains 2 million records.

In Fig. 1, the x-axis represents the order of records in the client input file. The y-axis represents the response time *speedup*, which is the time to process the batch of records on a serial computer divided by the time to process the batch of records on the grid of computers. Therefore, a larger number on the y-axis corresponds to a system with a lower response time.



**Figure 1: Response time speedup for a client input file with a distribution like the entire U.S. population with three different sorting methods**

As seen in Fig. 1, both the partitioning scheme that uses only zip codes and the partitioning scheme that uses zip code and last name perform well on client input files containing records that are sorted randomly. However, the scheme that partitions only using zip codes has a lower response time speedup value for a file sorted by zip code or sorted by zip code and last name because the load is not balanced across the grid computers. The partitioning scheme that uses the distribution of zip codes and last names has a higher response time speedup, which corresponds to a lower response time.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, we present two partitioning methods to load balance the workload of a database application across a grid of computers. Determining the distribution of the typical workload and partitioning the database to match it improved the load balancing and response time speedup. The methods used data about the U.S. population from publicly available data. Both proposed methods were able to handle random and sorted files with improvement of response time speedup.

Future work will include studying other distributions of the records to determine if this knowledge can be used to make better partitioning schemes. We would like to extend the idea of smaller partitions to workloads where the distribution of records is not known. It makes sense that a greater number of smaller partitions will aid in load balancing, but there is a greater amount of overhead that needs to be considered. In addition, we would like to develop methods to dynamically capture this data about the workload and use it to improve grid partitions.

For a grid system that has a known distribution of zip codes and last names, partitioning the database across the grid by zip code and last name is effective for decreasing the response time for a batch of records. The partitioning scheme that uses zip codes and last names only requires two tables and could be extended to a larger grid.

## 7. ACKNOWLEDGMENTS

Acxiom Corporation through the Acxiom Laboratory for Applied Research (ALAR) supported this research.

## 8. REFERENCES

- [1] Tannenbaum, A., van Steen, M. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Sadle River, NJ, 2002.
- [2] Jordan, H., Alaghband, G. *Fundamentals of Parallel Processing*. Prentice Hall, Upper Sadle River, NJ, 2003.
- [3] Fu, B. and Tari, Z. "A dynamic load distribution strategy for systems under high task variation and heavy traffic," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, Melbourne, FL, USA, Mar. 9-12, 2003, pp. 1031-1037.
- [4] Li, K.. "Deterministic and randomized algorithms for distributed on-line task assignment and load balancing without load status information," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, Atlanta, GA, USA, Feb. 27 – Mar. 1, 1998, pp. 613-622.
- [5] Lu, C. and Lau S. M. "A negotiation protocol for batch task assignments in dynamic load distribution," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, San Jose, CA, USA, 1997, pp. 447-453.
- [6] Anane, R. and Anthony, R. "Implementation of a proactive load sharing scheme," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, Melbourne, FL, USA, Mar. 9-12, 2003, pp. 1038-1045.
- [7] Tosun, A.S. "Replicated declustering for arbitrary queries," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, Nicosia, Cyprus, Mar. 14 – 17, 2004, pp. 748-753.
- [8] Chen, C.M. and Cheng, C.T. "From discrepancy to declustering: near-optimal multidimensional declustering strategies for range queries," *Journal of ACM*, vol. 51, no. 1, Jan. 2004, pp.46-73.
- [9] US Census Bureau. *Census 2000 Gazetteer Files*. <http://www.census.gov/geo/www/gazetteer/places2k.html>, 2000.