Comparing the Performance of Clusters, Hadoop, and Active Disks on Microarray Correlation Computations

Jeffrey A. Delmerico*, Nathanial A. Byrnes†, Andrew E. Bruno‡, Matthew D. Jones‡, Steven M. Gallo‡ and Vipin Chaudhary*

*Department of Computer Science and Engineering, University at Buffalo
†Netezza Corporation
‡Center for Computational Research, University at Buffalo

***Abstract*—Microarray-based comparative genomic hybridization (aCGH) offers an increasingly fine-grained method for detecting copy number variations in DNA. These copy number variations can directly influence the expression of the proteins that are encoded in the genes in question. A useful analysis of the data produced from these microarray experiments is pairwise correlation. However, the high resolution of today's microarray technology requires that supercomputing computation and storage resources be leveraged in order to perform this analysis. This application is an exemplar of the class of data intensive problems which require high-throughput I/O in order to be tractable. Although the performance of these types of applications on a cluster can be improved by parallelization, storage hardware and network limitations restrict the scalability of an I/O-bound application such as this. The Hadoop software framework is designed to enable data-intensive applications on cluster architectures, and offers significantly better scalability due to its distributed file system. However, specialized architecture adhering to the Active Disk paradigm, in which compute power is placed close to the disk instead of across a network, can further improve performance. The Netezza Corporation's database systems are designed around the Active Disk approach, and offer tremendous gains in implementing this application over the traditional cluster architecture. We present methods and performance analyses of several implementations of this application: on a cluster, on a cluster with a parallel file system, with Hadoop on a cluster, and using a Netezza data warehouse appliance. Our results offer benchmarks for the performance of data intensive applications within these distributed computing paradigms.[1]**

I. Introduction

The motivation for this investigation is the need for biostatisticians to perform correlation analysis on the data produced from Array Comparative Genomic Hybridization (aCGH, but also known as Chromosomal Microarray Analysis). In this technique, genes from the DNA of a cancer cell and a healthy cell are tagged with different fluorescent markers, and the ratio of their fluorescent intensities is calculated for each gene. The data produced represents the ratio of the gene copy number of the cancer cell to the control cell, a good indicator of the level of that gene's expression in the individual. Newer microarray machines are capable of measuring 244,000 or even 1,000,000 different locations within the genome. The data set initially produced is large, but not unwieldy. However, an important analysis of that data set is to measure the correlation of the value for each gene with the values of all the others. This method is applied in the investigation of the genetic causes of many types of cancer, for example [7,8,9,10,11]. For a microarray with N probes, this requires the calculation of an N-entry by N-entry correlation matrix. At double precision, such arrays can reach hundreds of gigabytes or more, prohibitively large sizes for desktop systems and even some shared high-performance computing resources. Aside from the storage requirements, the resources required to compute the arrays in a reasonable amount of time quickly advance beyond the capabilities of desktop systems and require high-performance systems. From this volume of data, only strongly correlated values must be extracted, which presents the need for systems with high throughput I/O. So in order to enable this analysis for the newer, higher resolution, microarray hardware, the computing systems need to be capable of high parallel scalability to compute the data quickly, high volume storage to archive the computed data, and high throughput I/O to scan the data quickly for important values.

We have implemented this data-intensive application in four different frameworks in order to compare their performance and scalability: on a cluster, on a cluster with a parallel file system, with Hadoop on a cluster, and using a Netezza data warehouse appliance. These four implementations were chosen in order to contrast the performance of a cluster using a SAN server with that of the same compute nodes using parallel and distributed file systems, as well as to compare the Active Disk approach embodied by the Netezza system with all three cluster implementations. We have developed software for computing the row-wise correlation, as well as performing queries against the correlation data for the retrieval of significant values within each of these computing paradigms. Although the ability of the end user to perform this analysis on a traditional cluster is limited only by the

[1]This research was supported in part by a grant from NYSTAR.

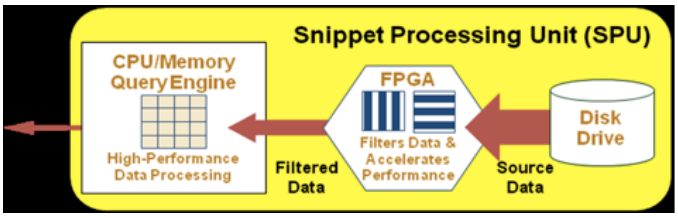

Fig. 1. The Snippet Processing Unit (SPU) includes a Disk, FPGA, general purpose processor and dedicated memory.

available hardware resources, the performance comparison indicates the need for a paradigm shift in the implementation of data-intensive applications such as this.

A. Mathematical Formulation

The Pearson product-moment correlation coefficient, or simply correlation, for two random variables represents the degree to which they are linearly related. For a pair of random variables (g_1, g_2), each with a mean μ, and standard deviation σ, it can be expressed as:

$$\rho_{g_1,g_2} = \frac{cov(g_1,g_2)}{\sigma_{g_1}\sigma_{g_2}}$$

where $cov(g_1, g_2) = E((g_1 - \mu_{g_1})(g_2 - \mu_{g_2}))$. The copy number ratio for each gene location in a data set is considered a random variable. Data consist of multiple measurements for each location, resulting in a two-dimensional data set of N rows of measurements (one for each gene location) of m samples each. For all of our implementations of this application, the correlation is calculated for every pair of gene locations in a data set using the following steps:

- Compute the mean (μ) and standard deviation (σ) for each row in the data set.
- For each measurement, calculate its deviation from its row mean.
- Multiply the corresponding deviations for the two rows of data.
- Find the mean of those products and divide by the product of the two rows' standard deviations.

Since we are considering pairwise correlation among N different genes, this results in an $N \times N$ correlation matrix if we compute all possible pairs. However, since $\rho_{g_1,g_2} = \rho_{g_2,g_1}$, only the upper triangle of the matrix contains distinct values. For all implementations, the degree of correlation does not affect the performance of either generating or querying the correlation matrix. The same number of computations must be performed to calculate each entry in the resulting correlation matrix, and a full scan of that matrix is performed, regardless of the correlation of the values.

B. Related Work

Unlike gene-sequencing applications such as BLAST (Basic Local Alignment Search Tool), to our knowledge, microarray correlation has not been implemented on FPGA-enabled hardware or on a cluster with Hadoop. Chilson describes parallelizing a similar correlation computation for a cluster,

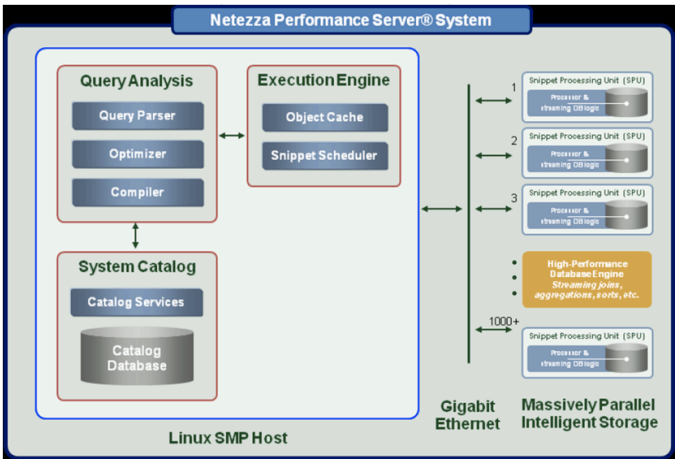

Fig. 2. The Netezza Performance Server System consists of a closely coupled server with many parallel Snippet Processing Units (SPU), each with their own disk, general purpose processor, dedicated memory, and FPGA.

but with a small data set [21]. There also exists an example of Interactive Supercomputing's Star-P software being used to run this application in parallel with MATLAB [22]. However, their demonstrated results are only for a small data set on a shared-memory machine which can hold the entire correlation matrix in memory. Since many biostatisticians prefer to work in R, there are several new libraries to enable distributed computations from R. SPRINT has been used on small problems from this domain [23]. RHIPE enables R users to perform computations with Hadoop, but no published examples exist for this particular application [24].

C. Hadoop

Apache Hadoop Core is a software platform for processing massive amounts of data efficiently across large clusters of commodity hardware [14]. Hadoop Core consists of the Hadoop Distributed File System (HDFS) and an implementation of the MapReduce programming model. The MapReduce algorithm specifies a Map function that processes a key/value pair and generates an intermediate key/value pair, a Reduce function then merges the intermediate key/value pairs down to the final output [15]. The Hadoop framework provides an API for creating MapReduce applications and a system for running these applications (referred to as MapReduce jobs) in a distributed environment over a cluster of fault-tolerant commodity machines. The framework controls the scheduling, monitoring, and execution of MapReduce jobs along with the management of HDFS storage nodes. A typical Hadoop cluster consists of a NameNode which serves as the master node for the HDFS [16], a JobTracker which serves as the central location for submitting and tracking MapReduce jobs [17], and a number of slave servers running both the DataNode/TaskTracker daemons [16].

A typical MapReduce job in Hadoop first splits the input data into a set of independent chunks which are processed in a parallel manner by the Map tasks (Mappers).

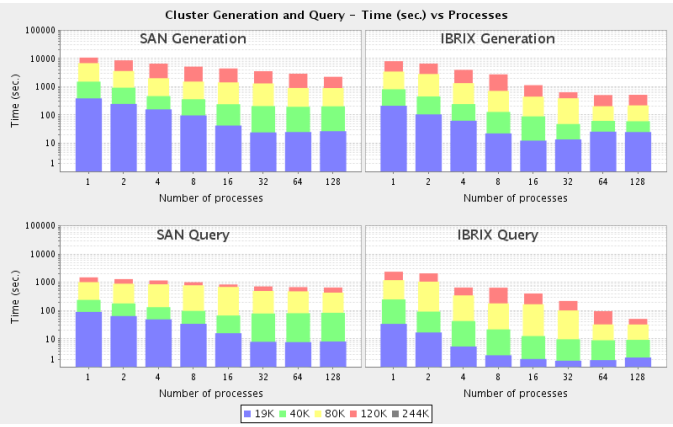

Fig. 3. Generation and Query times for both SAN and IBRIX

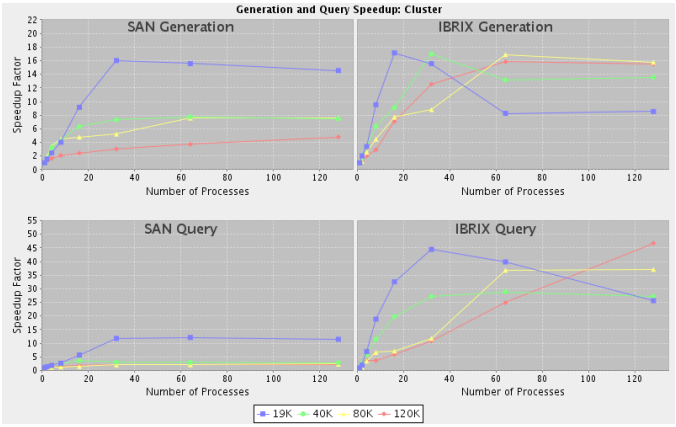

Fig. 4. Generation and Query speedup for both SAN and IBRIX

The output of each Map, in the form of key/value pairs, is then sorted by the framework and passed as input to the Reduce tasks (Reducers). The resulting output consists of one file per Reduce task [18]. There is also a combine operation (Combiners) which take place before the Map output is sent to the Reduce tasks. Combiners can work on the Map output in memory and perform a Reduce-type function [18]. This allows for fewer key/value pairs sent to the Reducer resulting in far fewer disk read/writes. The Microarray Correlation application defines a custom Combiner for use in computing the Top N values of the correlation matrix to decrease the amount of values sent to the single Reduce task.

D. Netezza Architecture

An effective approach to data-intensive computation is to integrate processing power and memory into a disk drive and allow application-specific code to be downloaded and executed on the data that is being read from or written to disk [1,2]. This alleviates the network bottleneck between the disk and the processor and this technology is commonly referred to as Active Disks [2]. To utilize Active Disks, an application is partitioned between a host-resident task and a disk resident task [1,12]. The objective is to offload processing to the disk-resident processors and to use the host processor primarily for coordination, scheduling, and combination of results from individual disks [1]. To accelerate the processing further, one can use special-purpose hardware to optimize certain kinds of operations. The Netezza Performance Server (NPS^{TM}) attempts to achieve this by using an array of Snippet Processing Units (SPUs), each of which is a disk attached to reconfigurable hardware, essentially an FPGA (Field Programmable Gate Array), and a microprocessor. By doing so, the special-purpose hardware can execute simple computations on the data while it is streaming from the disk (Fig. 1). Fig. 2 illustrates the overall structure of the NPS system.

The innovation and performance of Netezzas NPS system stems from the use of the FPGA, processor, and memory close to the disk. The FPGA is central to Netezza's FAST Engines framework, performing critical filtering and query processing functions as fast as data streams from the disk drive on each SPU. The typical impact of this work is a reduction of 95 percent or more in the data required for further processing by the on-board CPU and memory. As a commodity technology component, the FPGA is found in just about any "streaming" data product in the market today: from digital video recorders and DVD players to automotive electronics and displays to telecommunication switches to high-performance computing systems. As its name suggests, this small, low-power device is a highly reconfigurable, extensible functional element in the design of those products. Because of its widespread use the FPGA also enjoys a very robust technology curve, with projected five-year rates of price/performance enhancements that exceed those predicted for CPU technology by Moore's Law. The FPGAs are customizable at run-time to optimize their behavior for a particular snippet (program that is executed on an SPU). The use of FPGA close to the storage enables one to achieve two to three magnitudes of performance improvement on real applications in the areas of business intelligence analytics, bibliographical couplings, graph searches for sub-circuit motifs within integrated circuit netlists, and new word sense disambiguation approaches in natural language processing [4]. NPS has been applied to analysis of relationships in large data sets coming from heterogeneous data sources, a problem which proved intractable for traditional cluster hardware [3].

II. Methods

A. Cluster Approach

In order to parallelize this computation for a cluster, we leveraged a set of routines developed by several of the authors (Delmerico, Jones, Gallo) for working with large data sets. This software consists of several standalone applications as well as bindings for the R Statistical Computing Package which allow for the manipulation and analysis of large data sets which exceed the memory resources of the user. The standalone applications accept a text-based data set, in

this case the output of a microarray experiment, generate the correlation matrix, and then decompose it into smaller submatrices in order to manually "stripe" the data out over a storage array. These two steps are distinct in order to retain versatility for working with any large data set, not just our microarray correlation data. Lastly, a query routine scans the decomposed data and returns the values with the largest correlation. All of these steps are performed in parallel, with the work being divided into blocks of rows for each processor.

So for an $N \times N$ correlation matrix, each of the P processes computes the correlation of $\frac{N}{P}$ gene locations with all of the others, producing an $\frac{N}{P} \times N$ submatrix, the union of which contains all of the row-wise correlation values. Then each process decomposes its correlation submatrix into smaller submatrices and writes them out to disk. For a two-dimensional array of data, a block-cyclic decomposition is used. The user can specify the size of the tiles in order to optimize the performance for the system on which it is run. The output of the decomposition program is a group of files, each storing an individual tile from the decomposition, which are archived on disk so that they may subsequently be analyzed. During the decomposition, the dimensions and coordinates of each tile within the decomposition are calculated and output to a user and machine-readable file of meta-data. This file is later used during retrieval of the stored data. By decomposing this correlation matrix into many smaller, more tractable submatrices, and storing the meta-data about the decomposition, subsets of the entire matrix can be loaded into memory and analyzed or scanned individually. For the decomposition, which consists almost exclusively of I/O operations, the user can specify a buffer size which the program uses to aggregate reads and writes into a few large operations, rather than many smaller ones. This approach has been shown to provide increased performance in I/O-bound applications [13]. Lastly, each process scans the submatrix files from its portion of the overall matrix for the largest values, identifying the most correlated gene locations.

B. Hadoop Approach

The Microarray Correlation data set can be broken down into smaller independent sub-matrices for processing which makes it a good candidate for the MapReduce framework [14]. The implementation takes an $N \times M$ matrix of microarray sample data as input and subdivides the matrix row-wise based on the number of Map tasks. Each Map task processes a set of rows in the matrix generating the resulting $N \times N$ correlation matrix. Interestingly, this particular application contradicts a general assumption of the MapReduce model where it's assumed the output of Reduce is smaller than the input of the Map [20]. In the case of loading the Microarray Correlation data, the output of Reduce is the generated correlation matrix which is an order of magnitude larger than the input matrix passed to the Map.

The input matrix is represented as a binary file of 8-byte double values. The width and height of the matrix are also given as input. A pre-processing step first generates a temporary matrix by calculating, for each row, the deviation from the mean for each entry in the row. The temporary matrix is then written to the HDFS using Java's DataOutputStream for serializing primitive values. This file is then added to Hadoop's DistributedCache facility to ensure that the temp matrix file is copied to each node locally before the Map task is initiated. Prior to each Map tasks execution, a setup function is called which reads the temp matrix file from the DistributedCache and loads it into memory for use in computing the rows of the correlation matrix. The Mapper function is setup to receive an integer (i^{th} row of the temporary matrix) as the key and $Null$ for a value. There's no need to pass a value to the Map function as the temporary matrix has already been loaded into memory and contains all the necessary data for the computation. The Mapper function then computes the i^{th} row of the correlation matrix collecting the output into a key/value pair where the key is an integer (i^{th} row) and value is an array of doubles. No further reduction is needed in the Reducer step so the Identity Reducer is used to simply write the input key/value pair directly to the output. The output is written to a set of Hadoop SequenceFiles, one file per Map task. A SequenceFile is a flat file consisting of binary key/value pairs and is used extensively throughout the MapReduce framework for exchanging data between the Map and Reduce phases [19]. An example SequenceFile is of the format:

Key	Value
[row index]	[double double ... n]

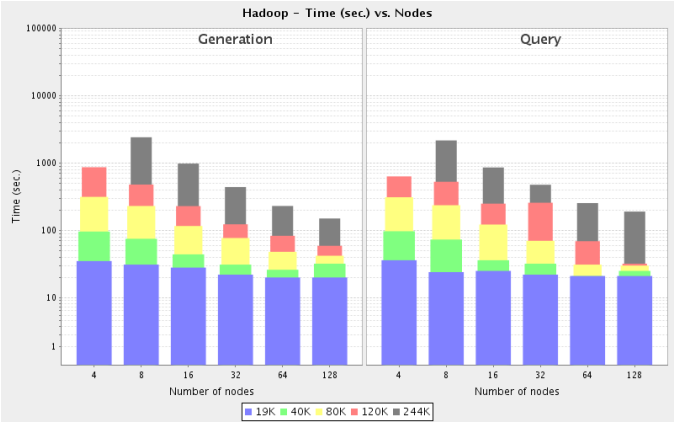

Fig. 5. Generation and Query times for Hadoop on the cluster

Pseudo code:

```
LoadMatrixMapJob:

function setup():
    matrix[][] = load temp matrix from DistributedCache

function map(int row, null):
    vector = double[]
```

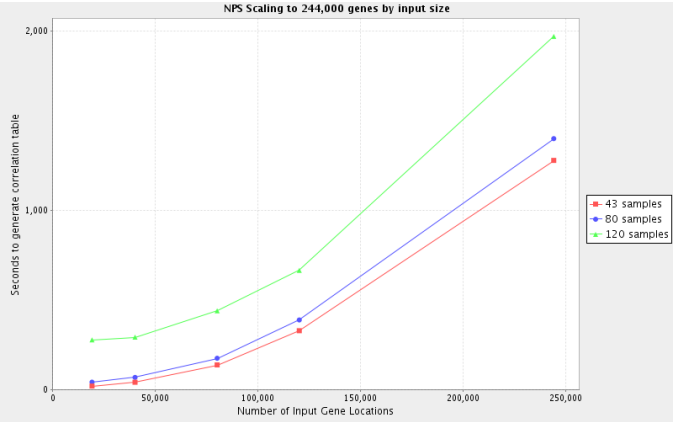

Fig. 6. Generation time for NPS by input size

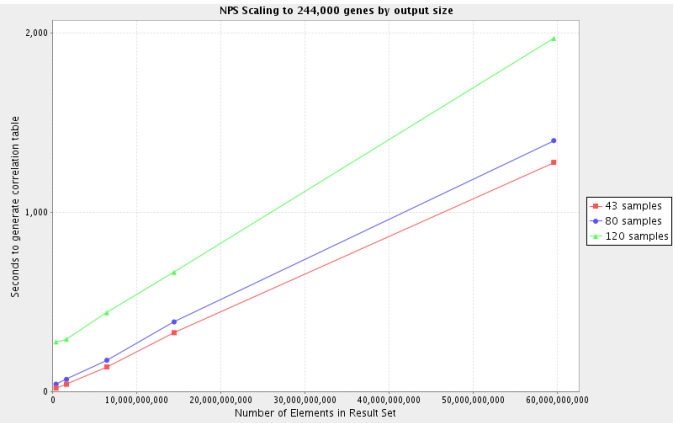

Fig. 7. Generation time for NPS by output size

```
for i = 0; i  matrix.length; i++ do
   for j = 0; j  matrix[i].length; j++ do
      vector[i] = matrix[row][j] * matrix[i][j];
   end for
end for
vector[row] = 1.0

output(row, vector)
```

To query for the top N values of the correlation matrix a Mapper, Combiner, and Reducer are implemented. Each Mapper task processes X rows of the correlation matrix collecting the top N values for each row. The output of the Mapper is a custom class encapsulating an entry, (i, j), and a value in the correlation matrix. The Combiner collects the top N values of the Mapper output passing the values off to a single Reducer task. The Reducer then computes the top N values of the output of the Combiner tasks writing the results to a file. The number of map tasks for the generation was set to equal the number of processors in the cluster (or 2 times the number of nodes). For searching the top N values, the number of map tasks was set to roughly the file size of the correction matrix divided by the HDFS block size, which was set to 128MB.

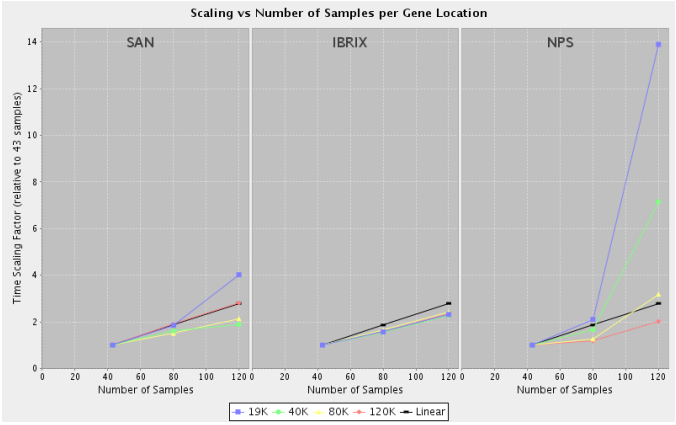

Fig. 8. Scalability by number of columns in the input data

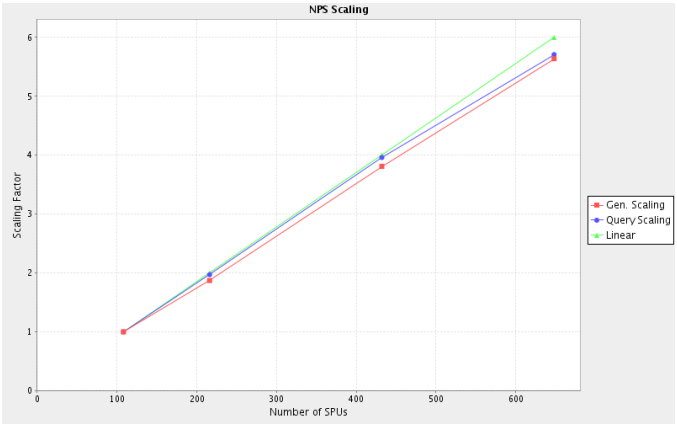

Fig. 9. Scaling of NPS systems of different sizes relative to single cabinet (108 SPU) performance.

C. Netezza Approach

As the Netezza system is a database system, we were able to simplify the solution to the microarray gene correlation problem to a set of SQL statements (queries) and a load of

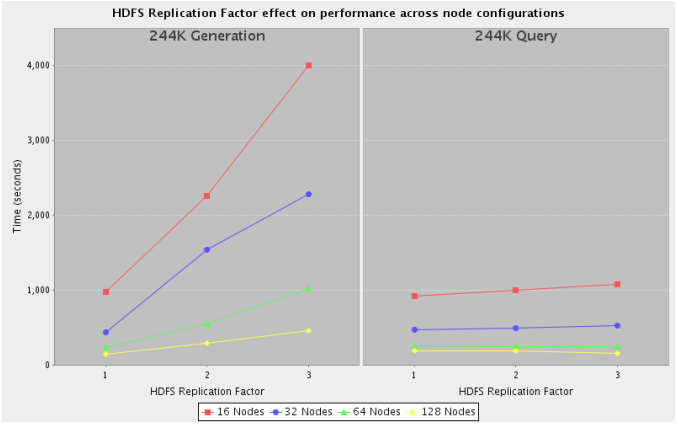

Fig. 10. Performance of Hadoop on 244k data set for several replication factors.

the data into the database. We generated synthetic data sets consisting of three columns of data: integer row and column indices and a random double precision floating point number. The integers represented the gene and sample numbers. In the case of the real data set, it was converted to the same three column format for standardization's sake. For example if there were 19,000 genes and 43 samples, the data file would have 19,000 x 43 = 817,000 records.

Once the original data was loaded into the database on the NPS 10400 system, three steps were performed:

1) Convert the 3 column table into a table with number of samples + 1 columns.
2) Compute the Standard deviation and mean for each row, and in the same step, calculate the difference from mean for each column and store the results in a temporary table.
3) Compute the Cartesian product of the temporary table with the restriction that the gene of the first reference is not equal to that of the second reference, or where the gene of the first reference is less than that of the second.

The output of this process is another database table with three columns: gene A, gene B, and the correlation. The final step in the testing process was to run an "interrogation" query against the result sets. The query chosen returned the 200 most correlated genes from the result sets. Any subsequent querying of the result set could be performed on an ad-hoc basis using SQL. The Netezza approach was implemented in less than a week by a single individual.

Brief Process Description:

1) Create raw data table with the columns of gene id, sample id, value.
2) Convert data into loadable format, or generate simulation data (ASCII text which matches raw data table structure)
3) Load data using Netezza Load Utility
4) Create a wide table with one row per gene, and one column per sample.
5) Create a work table that stores the sdev, mean, and variance from mean over each column, for each row.
6) Compute the correlation of every row to every other row in the work table through the use of a cross product SQL join of the work table with itself. The cross product was restricted with one of two conditions, the first ensured that we did not calculate the correlation of X to X, and the second to not compute the correlation of both X and Y and that of Y and X.
7) Query the 200 most correlated gene-gene sets from the result set as an example of result interrogation performance.

III. Results

The following performance tests were performed on the U2 cluster at the University at Buffalo's Center for Computational Research and on a Netezza NPS 10400 appliance. The U2

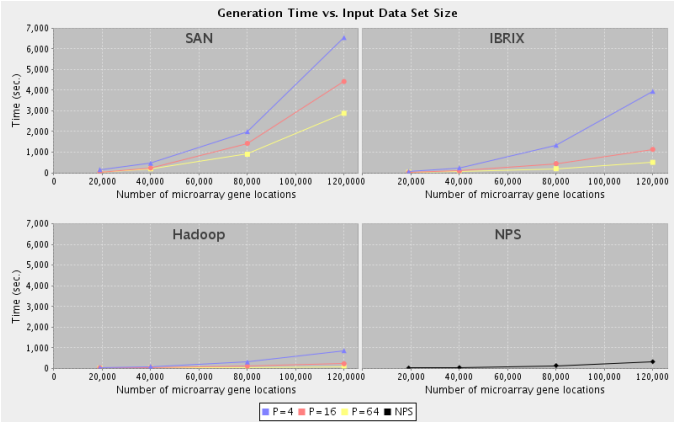

Fig. 11. Selected generation times by input size

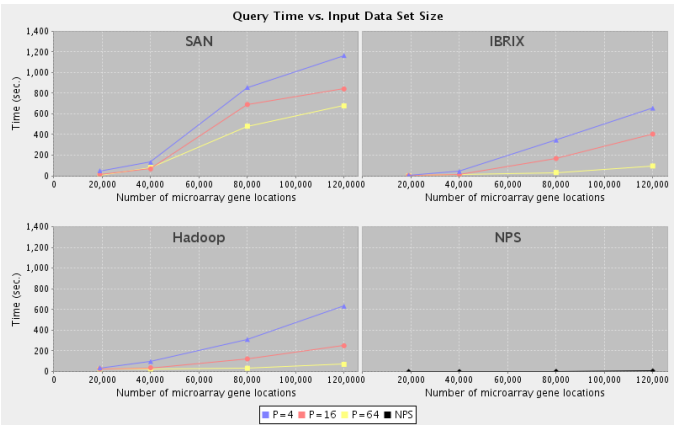

Fig. 12. Selected query times by input size

cluster consists of 1056 Dell PowerEdge SC1425 Servers, each with two Intel Xeon "Irwindale" processors at either 3.2 GHz or 3.0 GHz. Most of these nodes are equipped with 2 GB of local high speed memory, but several racks have 4 GB, and one rack has 8 GB. Each node has an 80 GB local disk and is connected to the rest of the cluster with either Gigabit Ethernet or both Gigabit Ethernet and Myrinet 2G. Of U2's two globally accessible NFS mounted storage arrays, one has roughly 2 TB of scratch space from a storage array network (SAN), while the other has approximately 25 TB of scratch space running an IBRIX parallel file system. Both the SAN and IBRIX scratch spaces were used during the cluster experiments, and the local disks for the participating nodes were used for the Hadoop runs as part of the HDFS distributed file system.

The Netezza NPS 10400 machine consists of four racks with 112 snippet processing units (SPUs) each, for a total of 448. However, 16 of the SPUs are used as "hot spares", so only 432 SPUs are applied to computations. Each SPU contains a 400 GB disk mated with an FPGA and a PowerPC chip, with the whole system capable of storing and operating on 50 TB of user data. All trials with the NPS 10400

were conducted using the entire machine for all database operations. The FPGA in each SPU only decompressed the data stored on disk as it was read; all other computation was performed by the CPUs on the SPUs. This application ran entirely on the SPUs, with the host only involved in the initial data load, and the management of the work on the SPUs.

In order to understand the implications of the following results, we must consider the differences in hardware between the cluster and NPS. After running some CPU benchmarks on both systems, we have determined that each core in one of our dual-core cluster nodes has 2.56 times the computational power of one NPS SPU. Therefore, the entire NPS 10400 system with 432 SPUs is equivalent to (432 SPUs * 1 cluster core/2.56 SPUs) $\approx$ 168 cores = 84 cluster nodes. Thus, 128 nodes of our cluster have roughly 1.5x the computational power of the NPS system on which we performed our tests, and the NPS falls between our 64 and 128 node cluster runs. Additionally, identical correlation tests were performed on NPS systems of several sizes: 1, 2, 4, and 6 racks of 108 SPUs each. The speedup relative to the single-rack system is illustrated in Fig. 9. Clearly, the performance of the NPS system scales in a nearly linear way with its size for both generation and query portions of the correlation tests.

Although the specifics of each implementation differ, they all followed the same high-level workflow: compute the correlation matrix, then decompose the matrix into smaller submatrices and archive on disk, then query the distributed data set for extreme values. For each data set, once the correlation matrix was computed, a query was performed to return the 200 most highly correlated results. In addition to the real-world data set of 19,116 gene locations with 43 samples at each, data sets of the following sizes were synthetically produced for both the cluster and the NPS. These sizes were chosen partially to provide estimates of scalability for both the number of gene locations and the number of samples at each, but also to approximate the data set sizes that would be produced from actual microarray experiments.

19,116 x 80, 120	80,000 x 43, 80, 120
40,000 x 43, 80, 120	120,000 x 43, 80, 120

The Hadoop trials included 19k, 40k, 80k, and 120k data sets for all processor counts, and 244k and 1M data sets for those in which the distributed file system had enough capacity. The generation times for trials on the cluster are shown in Fig. 3, for Hadoop in Fig. 5, and for the NPS in Figs. 6 7. The query times for the cluster can be found in Fig. 3, for Hadoop in Fig. 5, and for the NPS in the following table:

Data Set	Query Time (sec.)
19k	< 1
40k	1
80k	3
120k	7
244k	25
1M	489

Some additional generation times for Hadoop runs of various node counts and the NPS data sets beyond the capacity of the shared cluster storage are listed below (in seconds):

Hadoop				
Data Set	N=16	N=32	N=64	N=128
244k x 43	979	439	230	150
1M x 43				3765

NPS	
Data Set	Time(sec.)
244k x 43	1279
244k x 80	1400
244k x 120	1970
1M x 43	20343
1M x 80	22579

Although Hadoop offers many user-accessible parameters, the replication factor of the file system (dfs.replication) was of particular interest to us for its potential impact on performance. The default setting for this parameter is 3, meaning that three copies of all data would be distributed around the file system. For many applications where data would be resident on the file system for the long term, this level of redundancy may be necessary to prevent loss of data in the event of failures. However, since the volume of data of interest is small compared to the entire set of correlation values, and can easily and relatively quickly be re-computed, we are assuming that our data will be resident for only the short term; long enough to perform analysis and extract pertinent values. Therefore, our Hadoop trials were conducted with a replication factor of 1. However, the effect of the replication factor on performance remained of interest, so we performed further trials with a 244k data set to determine that relationship. In Fig. 10, the replication factor has little effect on the query portion but a more dramatic effect on generation. For larger node counts, it appears that the replication factor has a smaller impact on the performance at either task; to the point that it even improves the query performance slightly for 128 nodes. There is no clear tradeoff of larger generation times for smaller query times with greater replication. Thus the level of replication would likely be a decision for the end-user to make strictly based on a desired level of fault tolerance. Another parameter of interest was the number of map tasks, where a larger value may allow Hadoop to load-balance more effectively and recover more quickly from failures. However, further tests with a variable number of map tasks indicated that neither robustness nor performance was gained from increasing the cardinality beyond the number of processors in the job.

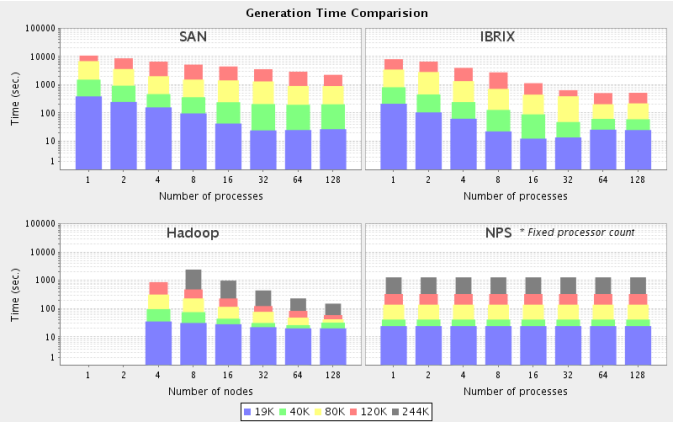

Fig. 13. Comparison of generation times across all implementations. Cluster and Hadoop times are relative to number of nodes, NPS times are for entire machine.

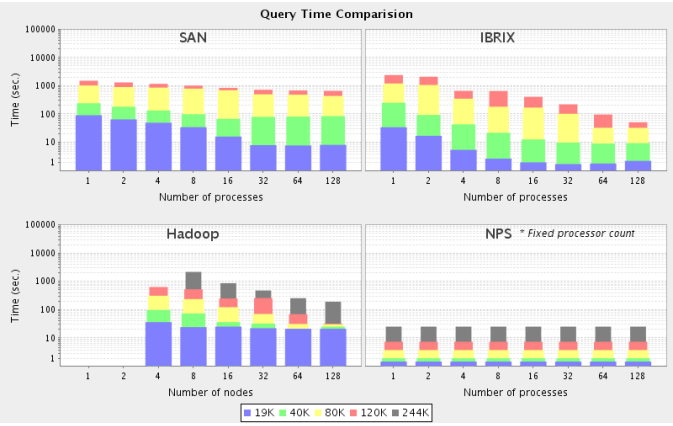

Fig. 14. Comparison of query times across all implementations. Cluster and Hadoop times are relative to number of nodes, NPS times are for entire machine.

IV. Analysis

It should be noted that the performance data for the cluster trials, specifically using the SAN for storage, represent a "best case scenario"; without a parallel file system, the time for I/O operations is heavily dependent on network traffic in and out of the storage array, so each trial was repeated multiple times and the lowest time selected. Any subsequent analyses comparing other implementations to these data should be regarded as a low estimate on the improvement in performance that one might attain.

In Fig. 4, we can see that the SAN trials are limited to single-digit speedups for all but the smallest data sets, and exhibit almost no improvement beyond 32 processes for either task. In contrast, the data from IBRIX scales almost linearly to 16 or 32 processes, but the performance either plateaus (for larger data sets) or drops off (smaller sets). However, for querying the 120k set, it continues to scale up to 128 nodes, but at only 30% efficiency. By contrast, the Hadoop implementation scales very well out to 128 nodes for larger problem sizes (Fig. 5). There appears to be some overhead for Hadoop which prevents additional performance gains for smaller data sets with large node counts, but even at these sizes, the Hadoop implementation still outperforms the others for generation (Fig. 13). The generation step scales very well on the the Netezza architecture (Figs. 6 and 7), although there is some overhead here as well for smaller data sets. The generation times for the largest data sets and the query performance from the tables in the previous section further indicate the excellent scalability on the NPS with respect to the size of the input data.

The overhead mentioned previously for generating the correlation matrix for small data sets on the NPS is more evident in Fig. 8, but for larger data sets, and for the other implementations, scaling with respect to the number of samples is fairly linear. In fact, since each sample represents a separate microarray experiment, possibly with a distinct patient, we would expect that for most analyses, $N_{samples} \ll N_{genes}$, and thus scaling in this dimension would likely be less of a concern than for the number of genes. In that respect, though, the computation time also appears to scale very well (Fig. 11). For a fixed number of processes in the cluster and Hadoop implementations, as well as with the NPS, the scaling for generating the correlation matrix appears to be the expected $O\left(N_{genes}^2\right)$. For the I/O-bound query portion, the scaling is not so uniform (Fig. 12). In this case, for implementations using the cluster, the parallel IBRIX file system and Hadoop both scaled well, much better than the SAN, but the NPS performs dramatically better than all of them at this task.

Considering the scalability of all four implementations relative to processor count (Fig. 13), the failure of both cluster variations to scale beyond about 32 nodes prevents them from competing well with either Hadoop or the NPS at the generation step. When leveraging at least 64 nodes, the Hadoop implementation is able to match or outperform the NPS for all input sizes. For the query portion of the trials, however, Netezza's active disk architecture clearly outperforms all of the implementations using the cluster resources, often by several orders of magnitude (Fig. 14). There appears to be some overhead here for the Hadoop version; despite its scalability, it requires considerably more time to return the top results than both cluster versions for small data sets.

Using the performance of the cluster implementation with the SAN as a baseline, Fig. 15 illustrates the relative speedup of the other implementations. For the cluster-based approaches, the comparison is made to the SAN data for the same number of processes or nodes. Since the NPS works as a single unit, the comparisons in that case are between the whole machine (432 SPUs) and the cluster trials with the various node counts. For all data sizes, the parallel file system adds some additional performance to the cluster

Fig. 15. Performance speedup relative to cluster generation trial with SAN, using same number of processes/nodes. In the case of the NPS, performance is for the whole machine (432 SPUs) relative to a variable number of processes for the cluster trials.

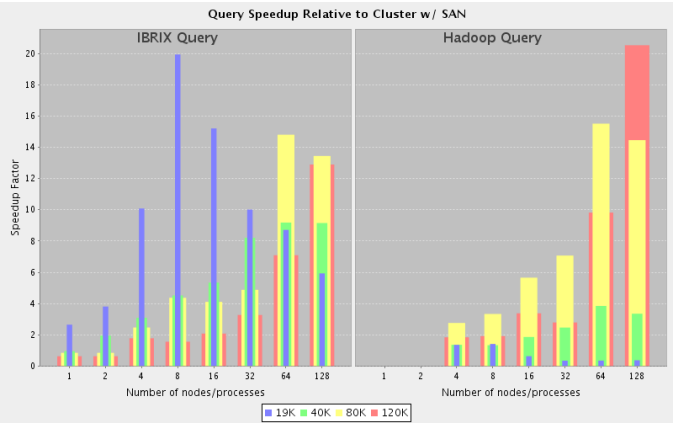

Fig. 16. Performance speedup relative to cluster query trial with SAN, using same number of processes/nodes.

implementation, but only marginally so. For small processor counts, the NPS performs at least an order of magnitude better than the cluster, but this performance is fixed, so the relative advantage decreases when compared to larger numbers of cluster nodes. However, because of the excellent scalability of the Hadoop implementation, its relative speedup grows with processor count, and ultimately outperforms all of the other versions at the generation task. As in Fig. 15, performance for the query task is displayed in Figs. 16 17 relative to the cluster with SAN. Here, although the IBRIX and Hadoop implementations add modest performance gains, clearly Netezza dominates. The NPS achieves a maximum speedup of 340x over one processor on an 80k data set, but also is able to maintain superior performance as the number of cluster nodes increases. The NPS exhibits 84x, 145x, and 94x speedups over 128-node queries on 40k, 80k, and 120k data sets, respectively.

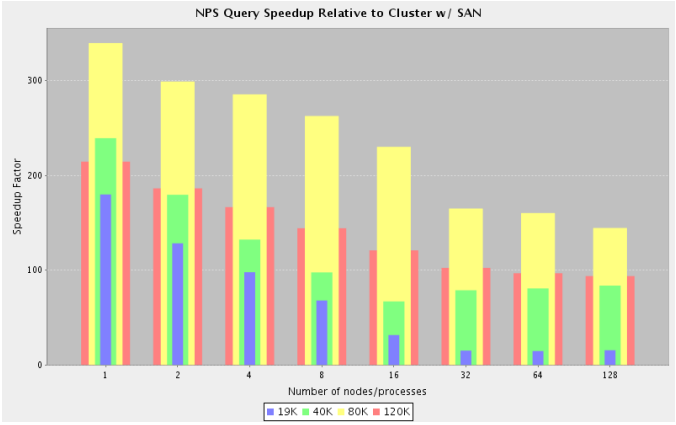

Fig. 17. Performance speedup relative to cluster query trial with SAN. Entire NPS machine (432 SPUs) compared to a variable number of processes for the cluster trials.

V. Conclusions

We have explored four implementations of an application intended to enable correlation analysis of microarray data, the volume of which is rendering it intractable on all but high-performance clusters and specialized hardware. Although the cluster versions demonstrate that this application can be implemented on more conventional hardware, the performance is lacking precisely because a cluster is designed for general-purpose computing. However, by leveraging the Hadoop distributed computing model, and its corresponding distributed file system, the hardware resources available in a cluster can be made to scale much better when applied to this type of data-centric application. Although Hadoop is able to work within the limitations of the cluster architecture to enable this scalability, a specialized machine such as the Netezza NPS is able to tackle these problems and scale them up beyond the capabilities of a cluster, and the performance can be significantly better, even for smaller problems.

In general, all four implementations scaled well in terms of the size of the input data set, both with the number of gene locations, and the number of samples at each. However, with the cluster implementations using shared storage resources, the input data sizes which can be operated on are limited by the resources available. The Hadoop implementation requires dedicated nodes in order to set up the distributed file system, and the storage resources are limited by the size of the local disks, but the performance benefits are significant, and with enough nodes, even large data sets can be computed. With 50 TB of storage capacity, the Netezza NPS 10400 offered no restrictions on the data set sizes which could be used as input within available microarray resolutions.

In terms of scalability and realistic performance, both Hadoop and Netezza display obvious benefits over the cluster implementations when working with very large data sets. Depending on the user's needs, as well as current and

prospective computing infrastructure, either one could offer the appropriate solution for performing these analyses of microarray data. Additionally, the development time also favors these two approaches; the cluster software took several months to develop from scratch and refine, while both the Hadoop and Netezza implementations each required only about a week of development and approximately two weeks to run our full complement of performance tests. The Hadoop implementation scales well for both tasks in this application, and performs the best in generating the correlation matrix. Although Hadoop did not perform the query portion as quickly as the NPS for the trials we attempted, and for some data sets, slower than the other cluster implementations, the scalability further indicate that this approach is still viable for queries and has the possibility of performing even better for larger node counts. Given a set of cluster resources, Hadoop is the best option for performing this analysis since it is a free software framework and can be easily implemented on the existing hardware. On the other hand, the NPS lags a bit in the generation step, but dominates in performing queries. These performance gains derive from the Active Disk architecture of the NPS, and for those users who can acquire a piece of specialized hardware such as this, the gains over a cluster can be dramatic as we have seen here. For those users with the need for specialized or additional hardware in order to perform this analysis, the Netezza NPS system may be the right solution.

Our goal in this work, while initially focused on comparing Netezza architecture with clusters, is not limited to Netezza hardware. Instead, the comparison sheds light on potential performance on the more generalized Active Disk architecture. Data-centric hardware systems have previously been proposed in the literature [5, 6]. They span a range of uses, from low power, minimal compute performance nodes [5], to more typical commodity rack-based systems [6]. Data-centric applications such as this have the potential to drive the development of this new computing paradigm forward, while simultaneously reaping the rewards of improved performance and extended capabilities in its application to the problem. These microarray correlation computations are but one of many data-centric applications that could be further enabled and expanded by a shift towards hardware which accommodates them.

REFERENCES

[1] A. Acharya, M. Uysal, and J. Saltz, "Active Disks: Programming Model, Algorithms and Evaluation", *Proc.Conf. Architectural Support for Programming Languages and Operating Systems*, IEEE Press, Piscataway, N.J., Oct. 1998, pp. 81-91.

[2] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle, "Active Disks for Large-Scale Data Processing", *IEEE Computer*, June 2001, pp. 68-74.

[3] M. Gokhale, "Disruptive Technologies: Field Programmable Gate Arrays", *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, June 14, 2007. http://www.csm.ornl.gov/workshops/SOS11/presentations/m_gokale.pdf

[4] G. S. Davidson, K. W. Boyack, R. A. Zacharski, S. C. Helmreich, and J. R. Cowie, "Data-Centric Computing with the Netezza Architecture", SANDIA REPORT, SAND2006-1853, Unlimited Release, Printed April 2006.

[5] D. Andersen, J. Franklin, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A Fast Array of Wimpy Nodes", Carnegie Mellon University PDL Tech. Report CMU-PDL-08-108, May 2008.

[6] A. S. Szalay, G. Bell, J. Vandenberg, A. Wonders, R. Burns, D. Fay, J. Heasley, T. Hey, M. Nieto-SantiSteban, A. Thakar, C. van Ingen, R. Wilton, "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing", Microsoft Technical Report MSR-TR-2008-187, Sep. 2008.

[7] M. Bredel, C. Bredel, D. Juric, G. R. Harsh, H. Vogel, L. D. Recht, B. I. Sikic, "High-Resolution Genome-Wide Mapping of Genetic Alterations in Human Glial Brain Tumors", *Cancer Research* 65, 4088-4096, May 15, 2005.

[8] H. Lee, S. W. Kong, and P. J. Park, "Integrative analysis reveals the direct and indirect interactions between DNA copy number aberrations and gene expression changes", *Bioinformatics* 2008 24(7):889-896.

[9] S. Bilke, Q. R. Chen, C. C. Whiteford, J. Khan, "Detection of low level genomic alterations by comparative genomic hybridization based on cDNA micro-arrays", *Bioinformatics* 2005 21(7):1138-1145.

[10] J. M. Nigro, A. Misra, L. Zhang, I. Smirnov, H. Colman, C. Grifn, N. Ozburn, M. Chen, E. Pan, D. Koul, W. K. Yung, B. G. Feuerstein, K. D. Aldape, "Integrated array-comparative genomic hybridization and expression array proles identify clinically relevant molecular subtypes of glioblastoma", *Cancer Research* 65: 16781686, March 1, 2005.

[11] J. A. Veltman, J. Fridlyand, S. Pejavar, A. B. Olshen, J. E. Korkola, S. DeVries, P. Carroll, W. L. Kuo, D. Pinkel, D. Albertson, C. Cordon-Cardo, A. N. Jain, F. M. Waldman, "Array-based comparative genomic hybridization for genome-wide screening of DNA copy number in bladder tumors", *Cancer Research* (2003) 63: 287280.

[12] E. Riedel, G. Gibson, C. Faloutsos, "Active Storage For Large-Scale Data Mining and Multimedia", *Proceedings of the 24th Conference on Very Large Databases*, August 1998.

[13] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, A. Sussman, "Tuning the Performance of I/O-Intensive Parallel Applications", *Proceedings of the Fourth ACM Workshop on I/O in Parallel and Distributed Systems*, May 1996.

[14] (04/09/2009). *Map/Reduce Tutorial*. Retrieved May 21, 2009 from http://hadoop.apache.org/core/docs/current/mapred_tutorial.html

[15] J. Dean, S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters", *Google, Inc.*, December, 2004. http://labs.google.com/papers/mapreduce.html

[16] (04/09/2009). *Cluster Setup*. Retrieved May 21, 2009 from http://hadoop.apache.org/core/docs/current/cluster_setup.html

[17] *Class JobTracker*. Retrieved May 21, 2009 from http://hadoop.apache.org/core/docs/current/api/org/apache/hadoop/mapred/JobTracker.html

[18] (05/20/2009). *HadoopMapReduce*. Retrieved May 21, 2009 from http://wiki.apache.org/hadoop/HadoopMapReduce

[19] (09/20/2008). *SequenceFile*. Retrieved May 21, 2009 from http://wiki.apache.org/hadoop/SequenceFile

[20] (05/20/2009). *MapReduce*. Retrieved May 21, 2009 from http://wiki.apache.org/hadoop/MapReduce

[21] J. Chilson, "Parallel Computation of High Dimensional Robust Correlation and Covariance Matrices." Master's thesis, Western Oregon University, 2001.

[22] Interactive Supercomputing. *Genomic Correlation at National Cancer Institute*. Retrieved August 28, 2009 from http://www.interactivesupercomputing.com/success/genomic_correlation.php

[23] *SPRINT: A new parallel framework for R*. Retrieved August 28, 2009 from http://www.r-sprint.org/

[24] S. Guha. *RHIPE - R and Hadoop Integrated Processing Environment*. Retrieved August 28, 2009 from http://ml.stat.purdue.edu/rhipe/index.html