**RC24188 (W0702-080) February 20, 2007**
**Computer Science**

# IBM Research Report

## Scalability of the Nutch Search Engine

**Dilma Da Silva, Parijat Dube, Maged Michael, José E. Moreira, Doron Shiloach, Li Zhang**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Scalability of the Nutch Search Engine

**Dilma Da Silva, Parijat Dube, Maged Michael, José E. Moreira, Doron Shiloach, Li Zhang**
**IBM Thomas J. Watson Research Center**
**Yorktown Heights, NY 10598**

**Abstract**

Nutch is an open source search engine that is gaining increasing popularity in the commercial world. The Nutch architecture leads itself to a wide range of parallelization techniques. Multiple back-ends servers can be used to both partition the corpus of search data, thus increasing the rate of queries serviced, and to increase the size of the search data while preserving the service rate. Alternatively, multiple search engines can operate in parallel, further increasing the query rate. In this paper, we analyze the performance and scalability of various configurations of Nutch. These configurations were implemented as part of the Commercial Scale Out project at IBM Research, and were used to investigate the applicability of scale-out architectures in commercial environments. We conclude that Nutch is highly scalable, with the different configurations behaving differently from a performance perspective.

## 1 Introduction

During the past 10 years we have witnessed the complete replacement of uniprocessor computing systems by multiprocessor ones. The revolution that started in the early to mid-eighties in scientific and technical computing finally caught up with the bulk of the marketplace in the mid-nineties. Today, the mainstream systems for commercial computing are what we call *scale-up* systems: large shared-memory systems that can be used for a variety of needs, from transaction processing to web serving.

More recently, there has been an increase in interest in clusters of loosely coupled machines for commercial computing. Each machine has its own private memory and the various machines communicate by passing messages over networks such as Gigabit Ethernet. These clusters are what we call *scale-out* systems. For many of the new web-based enterprises (e.g., Google, Yahoo, eBay, Amazon), a scale-out approach is the only way to deliver the necessary computational power. Also, computer manufacturers have made it easier to deploy scale-out solutions with rack-optimized and bladed servers. (Scale-out has been the only viable alternative for large scale technical scientific computing for several years, as we observe in the evolution of the TOP500 systems [15].)

We started the Commercial Scale Out (CSO) project at IBM Research to investigate how we could further promote the use of scale-out systems in commercial computing. Our work covers issues such as systems management, software distribution, system availability and also performance of emerging commercial applications, including search over unstructured data (*e.g*, web pages).

As a representative code of the operations involving search over unstructured data, we have chosen the Nutch search engine [1],[2]. Nutch is an open source search engine based on the Lucene framework [6] that is gaining increased popularity in the commercial world. A previous work [12] has reported a comparison between executing the Nutch search workload on scale-up versus scale-out systems. In this paper, we focus our study more on the scale-out behavior of Nutch. We show that there are many options for exploiting the inherent parallelism in search, and that each configuration has its own scalability characteristics. Whereas [6] reports only briefly on the scalability of Nutch, we here explore how both the number of back-ends and the data size per back-end affects performance and scalability. Furthermore, we report on results from multi-cluster configurations that represent yet another alternative for exploiting parallelism.

The rest of this paper is organized as follows. Section 2 describes scale-out systems, in particular the environment that we used for our studies, in more detail. Section 3 presents the Nutch workload that we ran in our CSO system. Section 4 reports our experimental findings, while Section 5 presents our estimates for how the system would behave at larger scales. Finally, Section 6 presents our conclusions.

## 2 Scale-out systems

The first form of scale-out systems to become popular in commercial computing was the rack-mounted cluster. These are local-area networks of independent servers, typically interconnected with conventional Ethernet networks. The servers themselves are optimized for high-density, since they can be rack-mounted, typically occupying only 1U to 4U of vertical space (1.75" to 7" high). There is really no hardware feature that distinguishes the servers in a rack-mounted cluster from individual servers. The only difference is the purpose with which they are used. Servers in a rack-mounted cluster are used as part of a system, performing specific functions that together compose the scale-out solution.

Examples of servers that are optimized for rack-mounting in the IBM product line include the p5 505Q (System p [11],[14]) and the xSeries 336 (System x). Both of these servers are 1U high and run standard software for either System p (AIX or Linux) or System x (Windows or Linux). These servers can be configured with up to 4 processors (using dual-core chips), and up to 16 GiB (x336) or 32 GiB (p5 505Q) of memory. Both servers include built-in support for Ethernet and SCSI, and two PCI-X expansion slots. Larger rack mounted systems are also available, like the 16-processor IBM p5 575 [9].

The IBM BladeCenter solution [4],[7],[8] (and similar systems from HP and Dell) represents the next step after rack-mounted clusters in scale-out systems for commercial computing. The blade servers used in BladeCenter are similar in capability to the densest rack-mounted cluster servers: 4-processor configurations, 16-32 GiB of maximum memory, built-in Ethernet, and expansion cards for either Fiber Channel, Infiniband, Myrinet, or 10 Gbit/s Ethernet. Also offered are double-wide blades with up to 8 processors and additional memory.

The differentiating characteristic of BladeCenter is the more integrated approach to building a cluster. Blades are assembled in a chassis that provide power, cooling, management infrastructure (through the Management Module) and networking (through switch modules) to the blades. Chassis connect to other chassis (or other non-bladed systems) and blades simply plug into the chassis, simplifying the physical deployment of additional blades to an existing cluster (assuming chassis slots are available).

In the Commercial Scale Out project, we chose BladeCenter as our platform for scale-out. Figure 1 is a high-level view of the CSO cluster hardware architecture. The basic building block of the cluster is a BladeCenter-H (BC-H) chassis coupled with a DS4100 storage controller through a 2-Gbit/s Fiber Channel. The CSO cluster consists of 8 chassis of blades (112 blades in total) and 8 DS4100 storage subsystems. The chassis themselves are interconnected through two nearest-neighbor networks. One of the networks is a 4-Gbit/s Fiber Channel network and the other is a 1-Gbit/s Ethernet network.

The BladeCenter-H chassis has 14 blade slots for blade servers. It also has space for up to two (2) management modules, four (4) switch modules, four (4) bridge modules[1], and four (4) high-speed switch modules. We have populated each of our chassis with two 1-Gbit/s Ethernet switch modules and two Fiber Channel switch modules.

Different kinds of blades were used in the CSO cluster. Approximately half of the cluster (4 chassis) is composed of JS21 blades. These are quad-processor (dual-socket, dual-core) PowerPC 970 blades, running at 2.5 GHz. Each blade has 8 GiB of memory, a 73 GB hard drive and a dual Fiber Channel adapter, which connects to the Fiber Channel switches in the chassis. The JS21 blades were the main focus of this study and the experiments reported in Section 4 are for those kinds of blades.

The DS4100 storage subsystem consists of dual RAID storage controllers, each with a 2 Gb/s Fiber Channel interface, and space for 14 SATA drives in the main drawer. Our drives have a raw unformatted capacity of 400 GB. Therefore, each DS4100 has a raw capacity of 5.6 TB. The drives in each DS4100 are configured as a RAID array and used to create LUNs of approximately 1 TB in size. After formatting and RAID redundancy is taken into account, each DS4100 can support 4 logical units (LUNs) of 1 TB each. Some smaller LUNs are used for various auxiliary purposes, so our storage system consists of approximately 30 LUNs of 1 TB each.

[1] Switch modules 3 and 4 and bridge modules 3 and 4 share the same slots in the chassis.

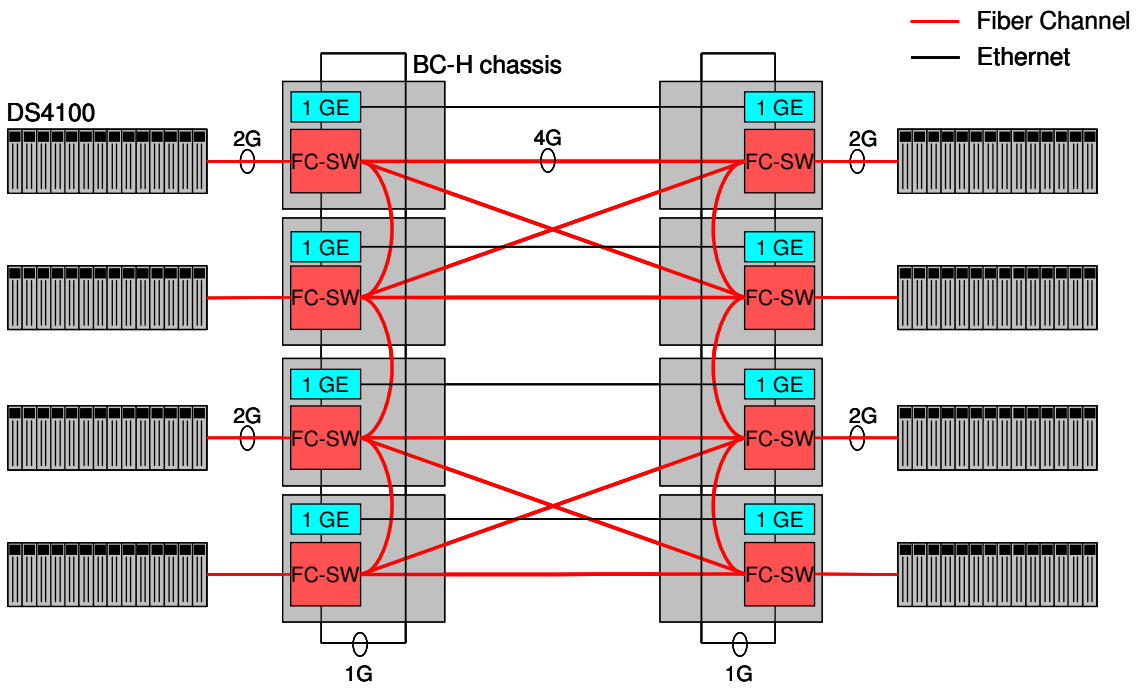



**Figure 1: Hardware architecture of the CSO cluster.**

Although each DS4100 is paired with a specific BladeCenter-H chassis, any blade in the CSO cluster can see any of the LUNs in the storage system, thanks to the Fiber Channel network. This all-to-all connectivity between servers and storage devices is central to our approach of using a GPFS shared file systems across all servers, as described below.

Each BladeCenter-H chassis in CSO contains two QLogic 20-port 4 Gb/s Fiber Channel switches. 14 of those ports are used to connect to the blades in the chassis. Each blade has two ports, each connecting to one of the switches. Of the additional six ports in each switch, one is used to connect to the DS4100 paired with that chassis. Each DS4100 has two ports, each connected to one of the switches in the chassis. Finally, the remaining ports are used to form a nearest-neighbor Fiber Channel network interconnecting all the corresponding switches in each chassis, following the topology shown in Figure 1. Since each chassis has two switches, we form two independent network planes with the same topology. For simplicity, Figure 1 shows only one of the planes.

GPFS (General Parallel File System) [13] is a parallel file system designed to support high I/O throughput as required, for example, in multimedia and HPC applications. GPFS has two main characteristics that make it scalable. First, it can stripe the files in the file system across multiple storage volumes (LUNs). This is independent and on top of any disk striping done by the storage controllers that implement the LUNs. Second, GPFS supports file caching in each node that implements the file system, with cache coherence between the nodes. This allows local caching of read/write data and greatly improves performance for those operations that can leverage the cache. Sequential reads/writes benefit from caching, and so do random reads/writes against working sets that (mostly) fit in cache, as is the case with Nutch query index data.

Figure 2 shows the high-level architecture of a system with GPFS. GPFS runs on all nodes that share that file system. In our CSO cluster, those nodes are the blades. We have built various independent file systems. In particular, the JS21 blades all share one GPFS file system. The nodes all plug into a switching fabric and on the other side of this switching fabric are the logical volumes (logical disks) that actually store the data for the file system. Not shown is the Ethernet network that interconnects all the blades and is used for file cache coherence traffic. The switching fabric is implemented by the Fiber Channel network and the logical disks are the LUNs discussed above. A single high-performance GPFS file system that can be shared by all blades allows a complete separation between servers and storage, since any server can see any file. That feature is critical in supporting our multi-cluster query operations (see Section 4).

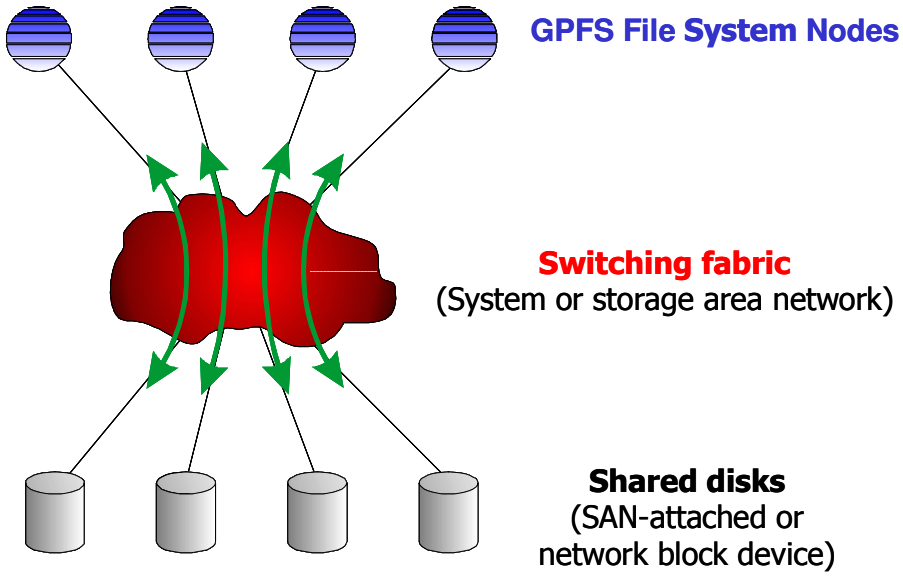



**Figure 2: General architecture of a GPFS system.**

## 3 Nutch

As previously mentioned, Nutch is a search application based on the Lucene framework. It is all implemented in Java and its code is open source. Nutch, as a typical search application, has three major components: (1) crawling, (2) indexing, and (3) query.

Crawling is the operation that navigates and retrieves the information in web pages, populating the set of documents that will be searched. This set of documents is called the *corpus*, in search terminology. Crawling can be performed on internal networks (Intranet) as well as external networks (Internet). Crawling, particularly in the Internet, is a complex operation. Either intentionally or unintentionally, many web sites are difficult to crawl. A lot of technology is required to develop a good crawler.

The performance of crawling is usually limited by the bandwidth of the network between the system doing the crawling and the outside world. In the Commercial Scale Out project, our laboratory setup did not have a high-bandwidth connection to the outside world and it would take a long time (months) to populate our system with enough documents to create an interesting corpus. Therefore, we decided to use a set of documents that had already been collected by the IBM WebFountain project [5], and did not further consider crawling further in our project.

Nutch includes a parallel indexing operation written using the *MapReduce* programming model [1]. MapReduce provides a convenient way of addressing an important (though limited) class of real-life commercial applications by hiding the parallelization and the fault-tolerance from the programmers, letting them focus on the problem domain. MapReduce was published by Google in 2004 and quickly became a de-facto standard for this kind of workloads.

A parallel indexing operation in the MapReduce model works as follows. First, the data to be indexed is partitioned into segments of approximately equal size. Each segment is then processed by a mapper task that generates the (*key*, *value*) pairs for that segment, where *key* is an indexing term and *value* is the set of documents that contain that term (and the location of the term in the document). This corresponds to the map phase, in MapReduce. In the next phase, the reduce phase, each reducer task collects all the pairs for a given key, thus producing a single index table for that key. Once all the keys are processed, we have the complete index for the entire data set.

In most search applications, query represents the vast majority of the computation effort. When performing a query, a set of indexing terms is presented to a query engine, which then retrieves the documents that best

match that set of terms. The overall architecture of the Nutch parallel query engine is shown in Figure 3. The query engine part consists of one or more front-ends, and one or more back-ends. Each back-end is associated with a segment of the complete data set. The driver represents external users and it is also the point at which the performance of the query is measured, in terms of *queries per second* (qps).

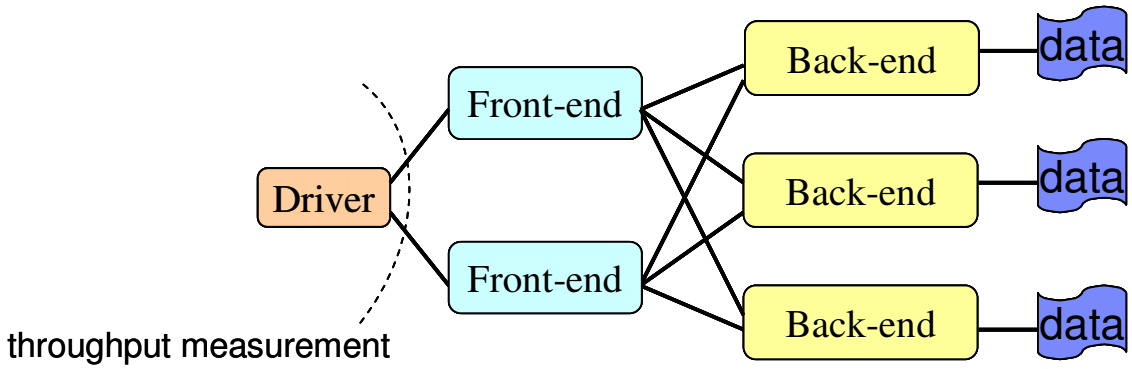



**Figure 3: Overall architecture of Nutch/Lucene query.**

A query operation works as follows. The driver submits a particular query (set of index terms) to one of the front-ends. The front-end then distributes the query to all the back-ends. Each back-end is responsible for performing the query against its data segment and returning a list with the top documents (typically 10) that better match the query. Each document returned is associated with a *score*, which quantifies how good that match is. The front-end collects the response from all the back-ends to produce a single list of the top documents (typically 10 overall best matches). Once the front-end has that list, it contacts the back-ends to retrieve snippets of text around the index terms. Only snippets for the overall top documents are retrieved, and the front-end contacts the back-ends one at a time, retrieving the snippet from the back-end that had the corresponding document in its data segment.

## 4 Experimental results

We start by reporting results from the experiments using the configuration shown in Figure 4. In this particular implementation of the architecture shown in Figure 3, there is one front-end running on a JS21 blade and a variable number of back-ends, each on their own JS21 blade. The data segment for each back-end is stored in an ext3 file system in the local disk of each blade.

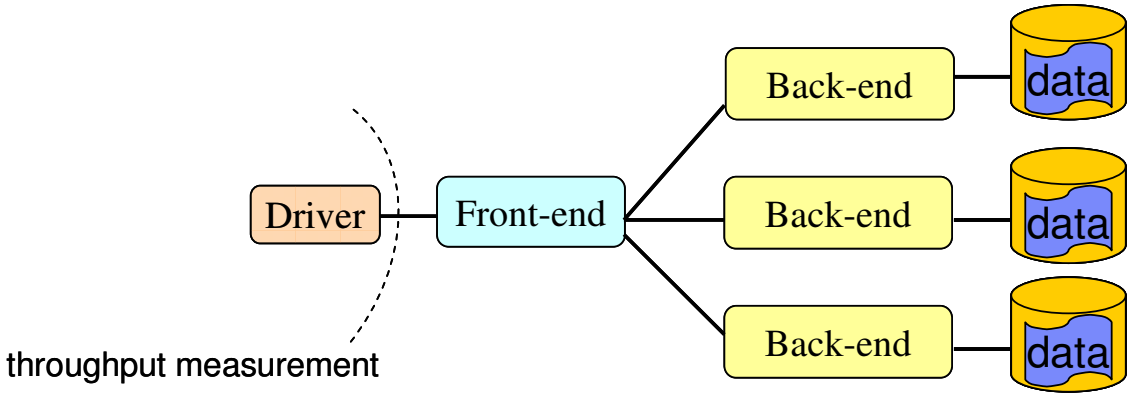
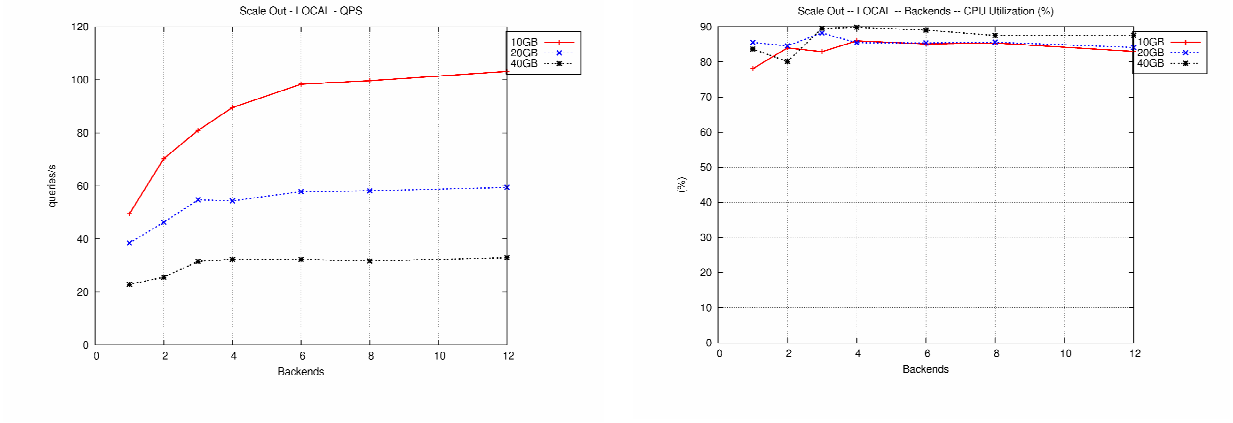



**Figure 4: Configuration with each data segment in an ext3 file system in the local disk of each JS21 back-end blade.**

Throughput measurements (queries per second) as a function of the number of back-ends are shown in Figure 5(a) for three different data segment size (per back-end): 10, 20, and 40 GB/back-end. The total data set size, therefore, varies from 10 GB (one back-end with 10 GB segment) to 480 GB (12 back-ends with 40 GB segment each). Figure 5(b) is a plot of the average CPU utilization in the back-ends as a function of the

number of back-ends. This latter plot shows that the CPUs are well utilized in this workload. (100% utilization corresponds to all 4 cores in the JS21 running all the time.)




(a) (b)

**Figure 5: Scalability results for query with data sets on local disk and ext3 file system. The plots show total queries per second as a function of number of back-ends (a) and average processor utilization in the back-ends as a function of number of back-ends (b).**

We observe in Figure 5(a) that the throughput, for a fixed data set size per back-end, increases with the number of back-ends. At first, this is a surprising result, since as we increase the number of back-ends, each query is sent to all the back-ends. We would expect a flat throughput or maybe even declining, as the front-end has to do more work.

We can explain the observed behavior as follows. Each query operation has two main phases: the search for the indexing terms in the back-ends (including the distribution and aggregation by the front-end) and the retrieval of the document snippets (including the requests from the front-end). In both of those phases, the work in the front-end is negligible for the size of systems we could experiment with. (See Section 5 for a prediction of how very large systems would behave.) The work per back-ends is constant with the number of back-ends for the first phase, but it actually decreases for the second phase. Since the total number of snippets retrieved is the same (10) independent of the number of back-ends, the more back-ends the less average work per back-end. This results in an increase in query throughput with the number of back-ends. The increase is less pronounced with larger data segment sizes because the back-end work in the first phase grows with the data size, but the work in the second phase does not change.

We also observe, in Figure 5(a), that query throughput decreases with the data segment size. This behavior is expected because a larger data segment results in larger indices and more document entries per index term. The decrease is less than linear with the increase in data segment size, so larger sizes are more efficient according to a queries/second * data size metric. The peak value for that metric, 15840 queries/second*GB, occurs with 12 back-ends and 40 GB/backend.

We conclude this discussion of query with data on local disks with plots of CPU utilization on the front-end (Figure 6) and network utilization for both front-ends and back-ends (Figure 7). We observe, in Figure 6, that indeed the workload on the front-end is small for these sizes of systems. The network utilization is also low, as shown in Figure 7, for both front- and back-ends.

The next configuration we experimented with is shown in Figure 8. The difference between this configuration and that shown in Figure 4 is that the data segments are stored in the globally accessible GPFS file system. We still assign each segment to one back-end, but that assignment no longer depends on the placement of the data in a particular storage device. Any blade in our CSO customer can be assigned to any of the data segments.

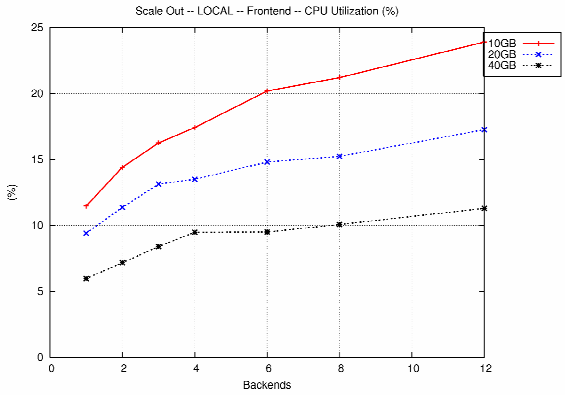



**Figure 6: CPU utilization in the front-end as a function of the number of back-ends.**

Throughput measurements (queries per second) as a function of the number of back-ends, for the configuration in Figure 8, are shown in Figure 9(a) for three different data segment size: 10, 20, and 40 GB/back-end. Figure 9(b) is a plot of the average CPU utilization in the back-ends as a function of the number of back-ends. We note that the behavior with GPFS varies with the data set size in a way similar to what we observed in Figure 5(a). However, the actual rates achieved with GPFS are 3 to 4 times lower. These measurements indicate that some optimizations are still necessary with GPFS.

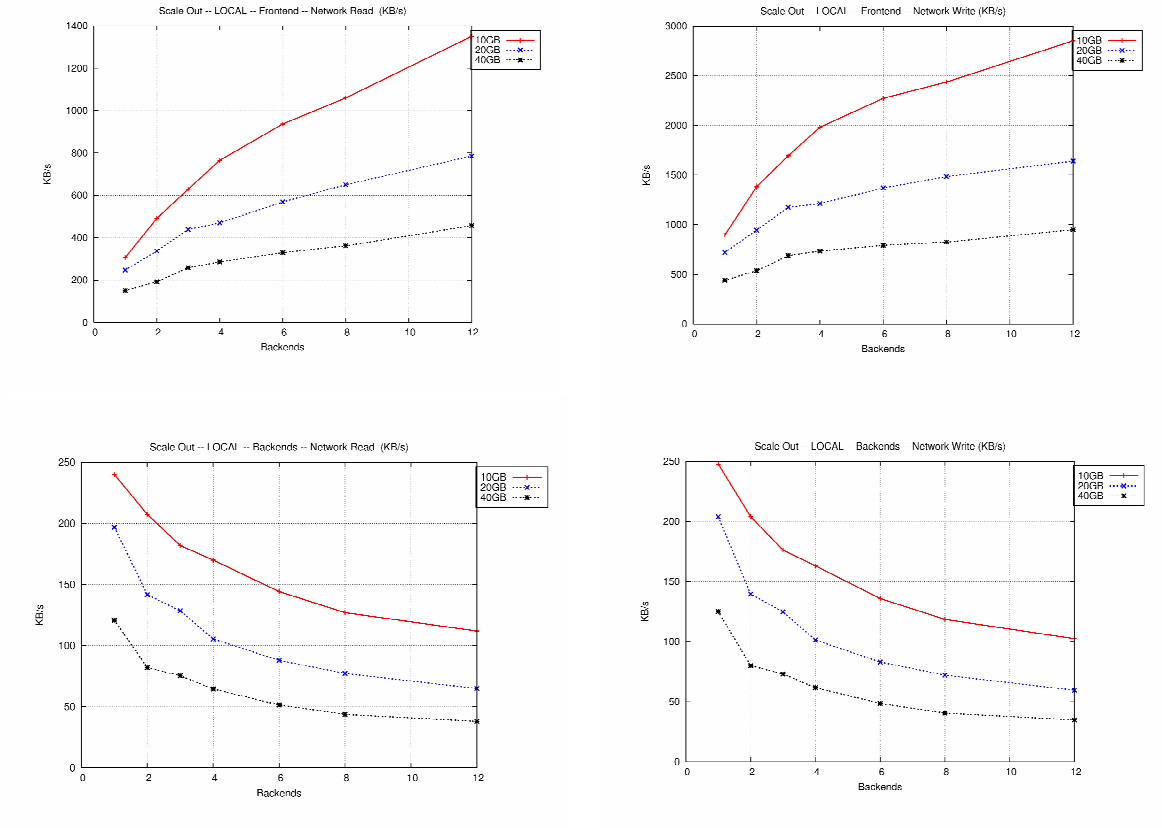



**Figure 7: Network utilizations in the front-end and average for back-ends, both read and write.**

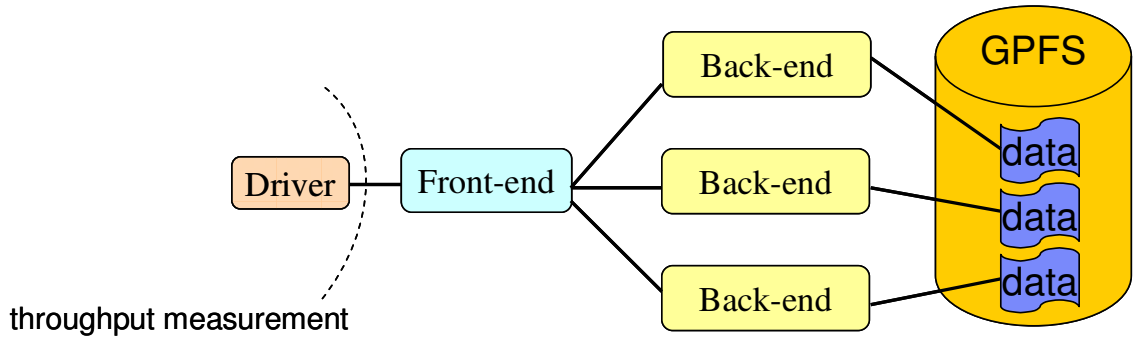
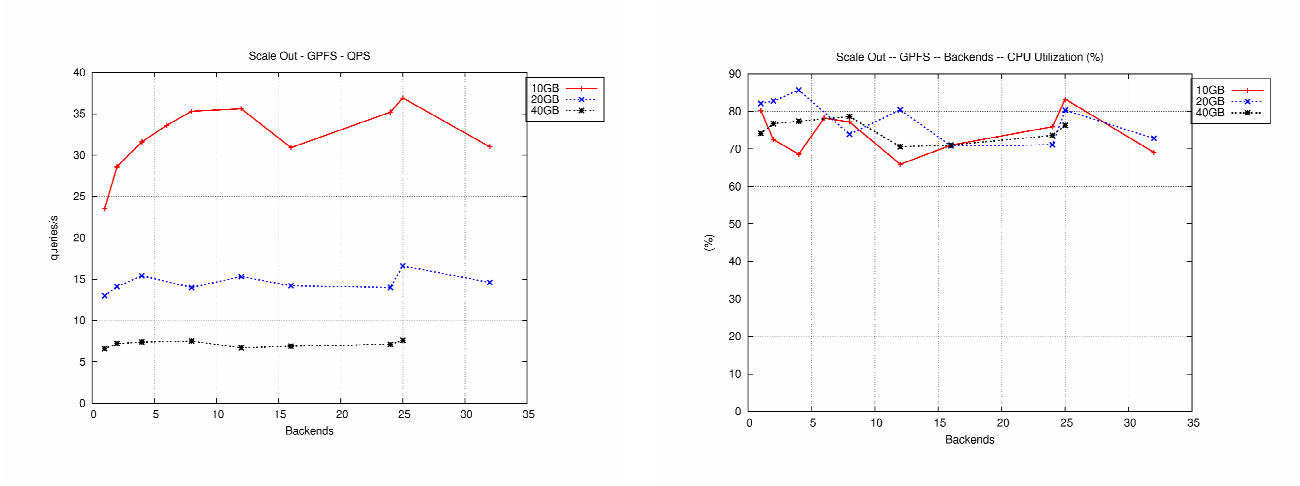



**Figure 8: Single-cluster GPFS configuration.**




(a) (b)

**Figure 9: Scalability results for query with data sets on the GPFS file system. The plots show total queries per second as a function of number of back-ends (a) and average processor utilization in the back-ends as a function of number of back-ends (b).**

Additional scalability experiments were performed using the configuration shown in Figure 10. We call that configuration multi-cluster. The back-ends are organized into multiple clusters. Each cluster has its own front-end and the number of back-ends is the same for all clusters. Each back-end in a cluster is responsible for a data segment and the data segments are all stored in a globally accessible GPFS file system.

Throughput results for the multi-cluster configuration are shown in Figure 11. We observe, in general, good scalability with the number of clusters. Using the metric of queries per second times the data set size, the best result occurs for 3 clusters of 12 blades each and a data set size of 10 GB per blade (10,536 queries per second * GB).

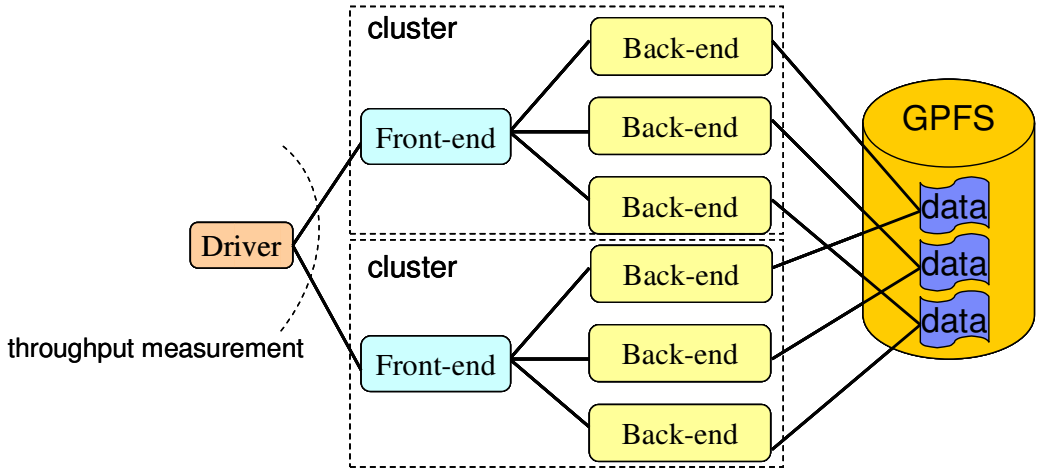



**Figure 10: Multi-cluster query configuration. The data resides in a globally accessible GPFS file system. Any back-end can, in principle, access any data. The back-ends can be organized into multiple clusters, each with its own front-end.**

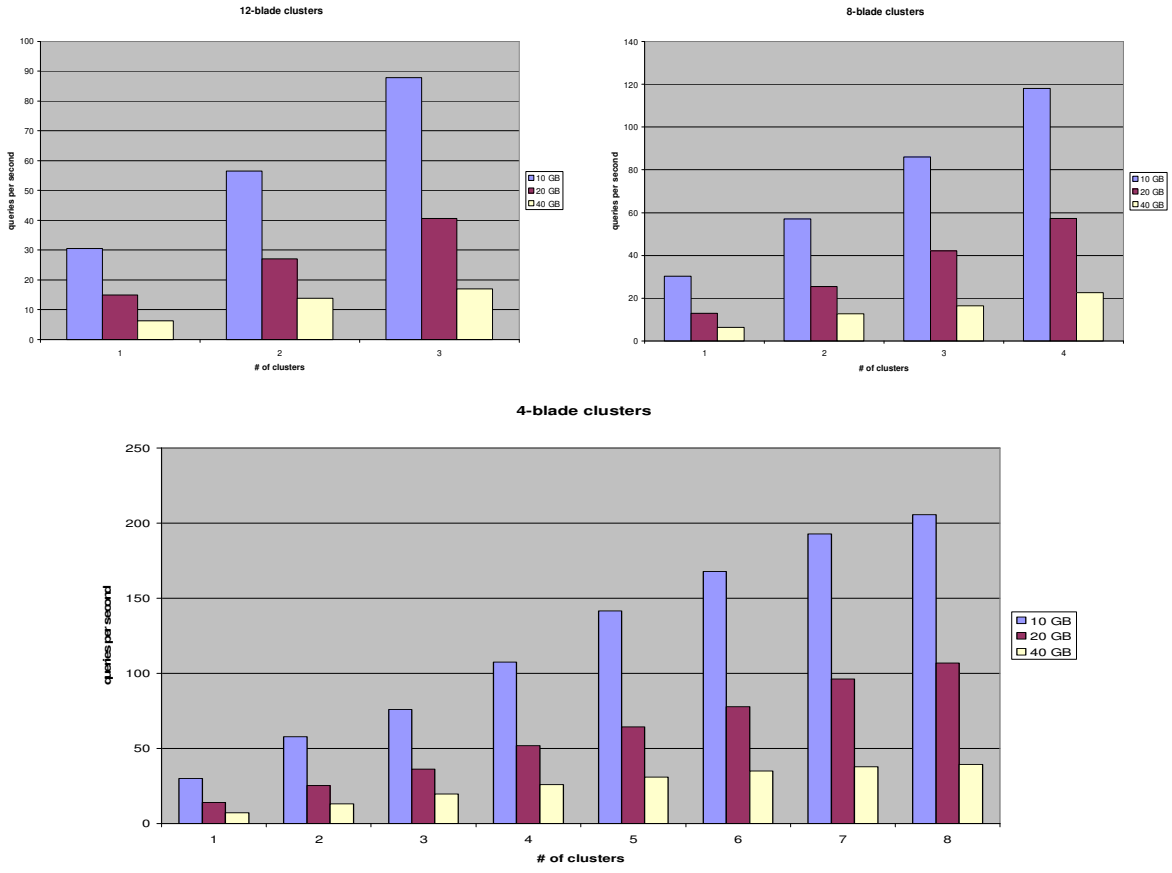



**Figure 11: Throughput results for multi-cluster configuration. Each plot is for a particular cluster size. In each plot, we show results for three different data sizes (per blade): 10, 20, and 40 GB.**

# 5 Performance projections

Although intuitive arguments can be made about scalability of our workloads beyond the system sizes we can directly measure, we wanted to perform a more quantitative assessment. For that purpose, we developed a modeling approach to investigate the behavior of query operations for search at large scale.

Performance projections for the Nutch query operation were performed using an innovative methodology combining queueing theory, statistical regression and non-linear constrained optimization. The approach is based on the techniques employed in the AMBIENCE toolset developed at IBM Research [9],[18]. We next describe the main steps of our methodology.

## 5.1 Model Abstraction and Analysis

We start by abstracting the experimental set-up as a queueing network model, as shown in Figure 12. There are two kinds of servers in the network (front-end and back-end) and three kinds of delay components (driver network, front-end network, and back-end network). Think time at the driver is zero, so there is no delay there. This is a closed queueing network, with a finite population of size equal to the number of outstanding query requests from the driver.

An important difference between our system and other applications modeled in [9],[18]is in the nature of the interaction between the front-end and back-end. As discussed earlier in Section 3, when a front-end receives a query from the driver it sends requests to all the back-ends and then waits for replies from all of them. Analysis using simple closed queueing network models, assuming load balancing among back-end servers, as done in earlier works [9] will fail to capture this inherent dynamics of our system. We modeled this interaction using *fork-join service stations* as shown in Figure 12. We use *Mean Value Analysis* for closed queueing networks with fork-join systems [16] to obtain analytical expressions for the performance metrics (end-to-end response time, serve utilizations, network utilizations) in terms of the model parameters (population size, service times at front-end and back-end servers, network delays).

### 5.2 Model Calibration and Validation

We then use measurements from real experiments to solve an optimization problem and determine the best-fit for the five unknown queueing network parameters (service time for the two kinds of servers and the three network delay components) that minimize the difference between measured data and results from solving the queueing network. The service time at the network delay components is expressed in terms of amount of network read/write data in Kilobits per query (Kb/q). We first describe our measurement data.

The measurements come from a large number of experiments with different configurations identified by the data set size per backend, the number of back-end servers and the population size. All experiments are done with a single front-end server. The following configurations are used:

- Data set size per back-end: 10GB, 20GB, and 40GB.
- 1 front-end server
- Number of back-end servers: 1, 2, 4, 8, 12, 16, 24, 32
- Population size: 10 and 60, for 1, 2, 4, 8 back-end servers
- 60 for 12, 16, 24, 32 back-end servers

And for each configuration, the following performance metrics are measured:

- CPU utilization of front-end and back-end servers
- Front-end and back-end network utilization
- Average response time

We identify each experimental configuration by the triplet: (partitions size in GB, number of back-end servers, population size). Thus we have measured data for thirty-six different configurations. Next for each experimental configuration we infer the five unknown parameters.

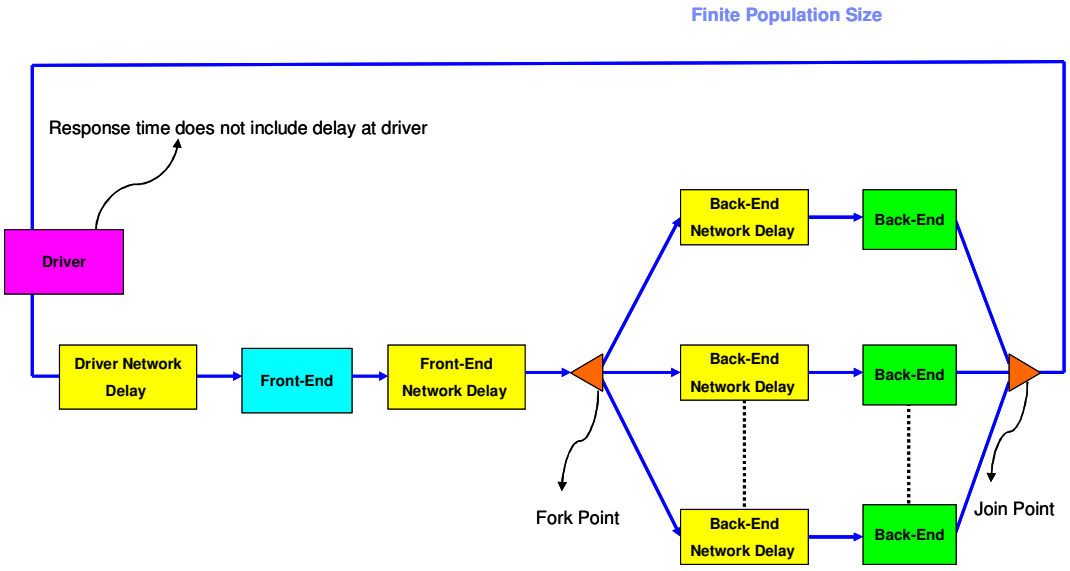



**Figure 12: Queueing model for Nutch/Lucene query.**

We formulate a constrained optimization problem with the objective being to minimize the difference between the measured values of the performance metrics and their analytical expressions from the model and with constraints on the difference between measured and analytical values. This allows us to limit the feasible values of service times to those which are compliant with our measured data. The solution to this optimization problem gives us the service times at the two servers and at the three network components.

Since the service times at servers and network components are invariant to population size we can use the model calibrated using inferred values for population size 10 to predict the performance for population size 60 and thus do a validation check on our model. Figure 13 shows the predictions for configuration (40GB, 8 back-end, population 60) using the model calibrated with configuration (40GB, 8 back-end, population 10). We observe that performance predictions from our model are quite close to the measured values for all the seven performance metrics (response time, throughput, front-end and back-end server utilizations, front-end, back-end and driver network utilizations).

We also observe that the service times at different components are sensitive to both the data set size per back-end and the number of back-end servers. In order to use our model to predict performance for large-scale configurations we need to understand the dependency of the service times on these configuration dependent parameters. We next employ statistical regression techniques and express service times as functions of the data set size per back-end and the number of back-end servers.

We use the inferred values of service times from the following 20 configurations for *multivariate regression* [17]: (10 GB, {1, 2, 4, 8, 12, 16, 24} back-end, population 60); (20 GB, {1, 2, 4, 8, 12, 16, 24} back-end, population 60); (40 GB, {1, 4, 8, 12, 16, 24} back-end, population 60). Figure 14 shows the inferred service times and the regression hyperplane. The analysis characterizes service times at different components as functions of the data set size per backend and the number of back-end servers.

Table 1 shows these estimated functions and the corresponding goodness of fit coefficient $R^2$. This coefficient is a statistical measure for regression and takes values between 0 and 1. The closer the value is to 1 the higher is the accuracy of the corresponding function. We also define an alternate measure of the closeness of hyperplane to the measured data values, the average relative distance from hyperplane as:

$$\Delta = \frac{1}{N}\sum_{n=1}^{N} \frac{|\,\text{Inferred Service Time(n)} - \text{Regression Estimated Service Time(n)}\,|}{\text{Regression Estimated Service Time(n)}} \text{X } 100\,,$$

where *N* is the number of data points (20) and *(n)* refers to the values for the *nth* data point.

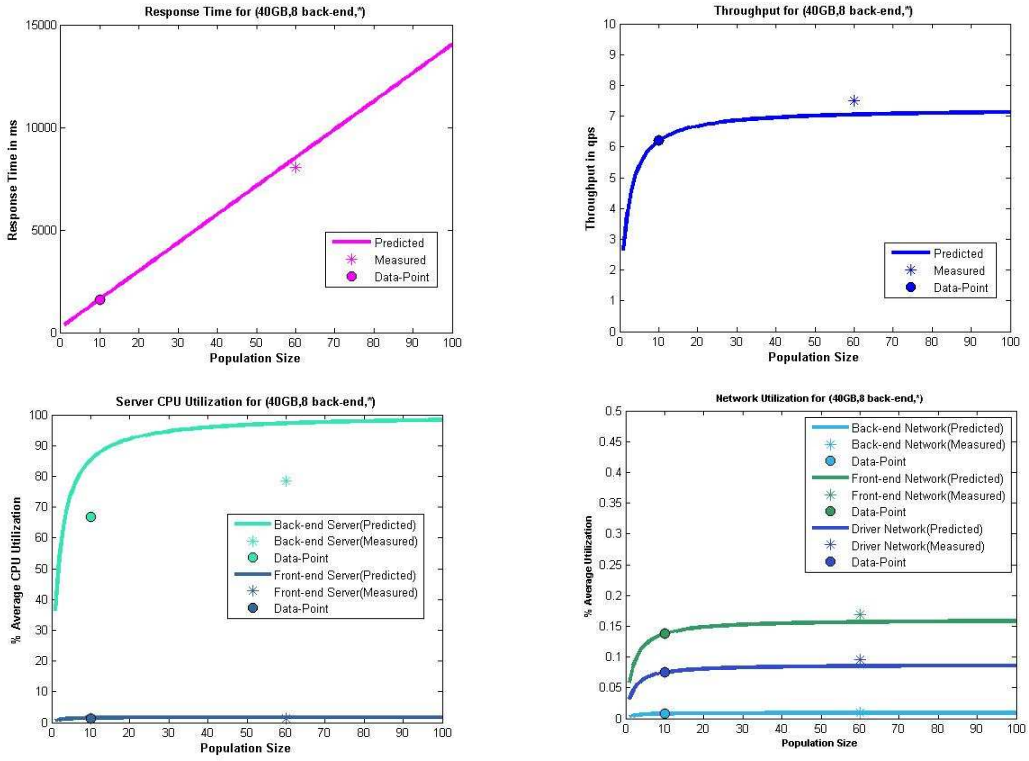



**Figure 13: Performance prediction for (40GB, 8 back-end, population 60) using model calibrated from (40GB, 8 back-end, population 10).**

**Table 1: Regression estimates capturing the dependency of model parameters on the data set size per back-end and the number of back-end servers.**

| Parameter | Regression Function | $R^2$ | $\Delta$ |
|---|---|---|---|
| front-end server (ms) | 2.457-0.020*d+0.063*b | 0.515 | 10.68% |
| back-end server (ms) | -7.927+3.590*d+14.069*(1/b) | 0.988 | 3.99% |
| front-end network read (Kbq) | 127.261+1.160*d+6.031*b | 0.988 | 1.65% |
| front-end network write (Kbq) | 34.879+0.609*d+5.087*b | 0.982 | 3.44% |
| back-end network read (Kbq) | 4.169+0.049*d+38.328*(1/b) | 0.989 | 2.32% |
| back-end network write (Kbq) | 4.158+0.101*d+36.156*(1/b) | 0.986 | 3.91% |

From the equations in Table 1, we observe that the service time at front-end increases linearly with the number of back-end servers, which is intuitive as more back-end means more work for the front-end which needs to send and receive data from all the back-ends. We also observe that the service-time at a back-end server however decreases with the increase in the number of back-end servers. This is conformant with the explanation we provided in Section 4 regarding our experimental observation for the increase in throughput with the number of back-end servers.

Having obtained these relations we can now make predictions of the behavior of the system for different configurations with varying data set size per back-end and varying number of back-end servers. Those predictions are summarized next.

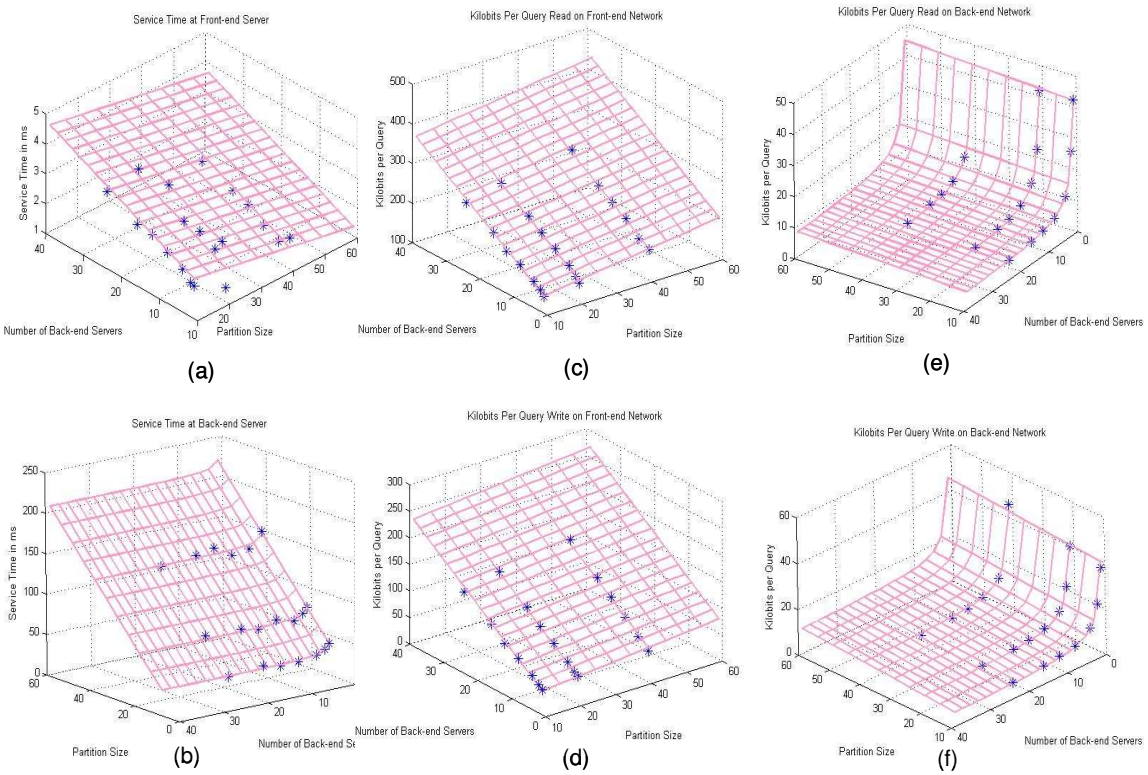



**Figure 14: Regression fit curves (a) front-end server service time (b) back-end server service time (c) front-end network read per query (d) front-end network write per query (e) back-end network read per query (f) back-end network write per query.**

## 5.3 Model Based Performance Prediction

Figure 15 shows model predictions for query in a cluster consisting of one front-end and a varying number of back-end blades. The markers indicate measurements, while the solid lines represent the predictions of the model. We observe a generally good agreement between measurement and prediction for small number of back-end servers. We also observe that the response time is essentially flat with the number of back-end servers, up to 500, 1000, and 2000 servers for data set sizes (per server) of 10 GB, 20 GB, and 40 GB respectively. The system is more scalable for larger data set sizes because the work per back-end server per query increases. This means that other parts of the system take longer to saturate.

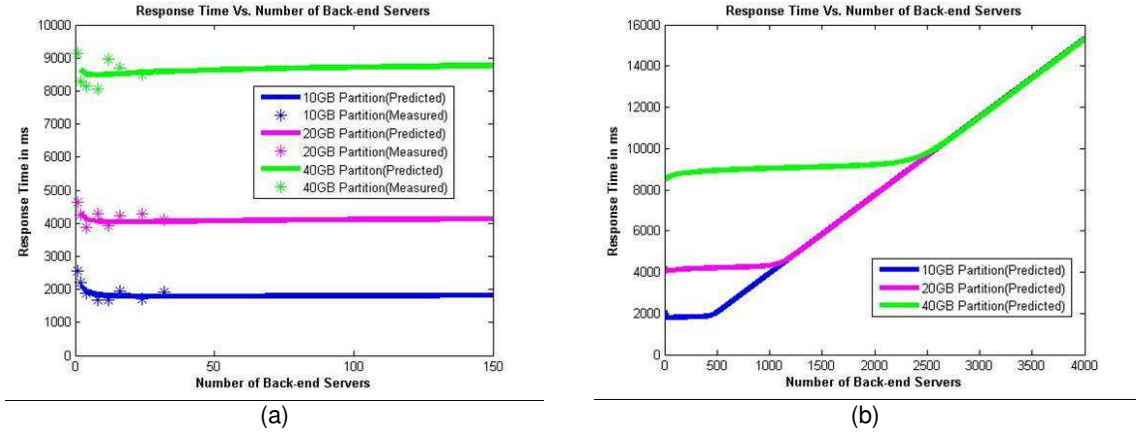
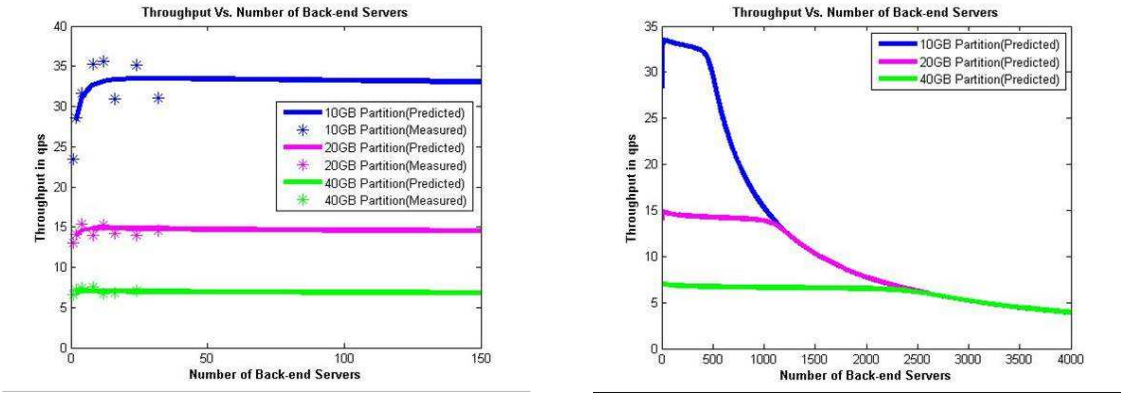



**Figure 15: Response time for query as a function of the number of back-end blades.**

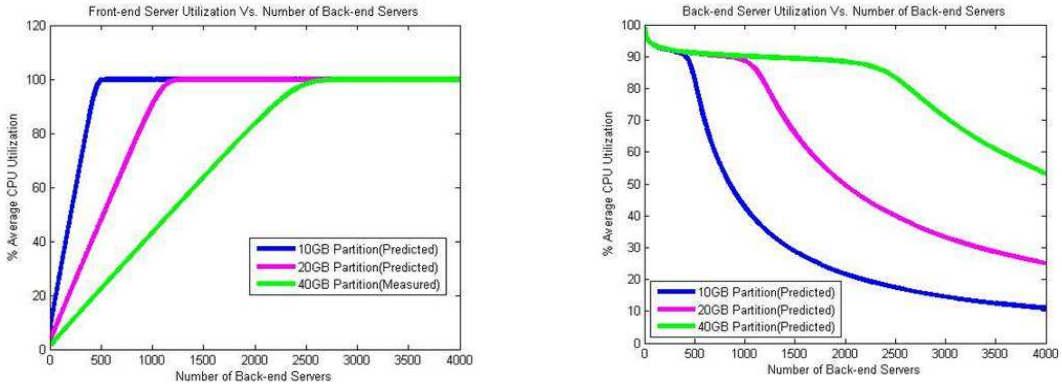
Figure 16 shows the corresponding predictions for throughput. Since throughput and response time, for a fixed population, are connected by the Little's formula (population = throughput * response time), the behavior of throughput is similar to that of response time. Specifically, throughput is more or less flat for up to 500, 1000, and 2000 back-end servers for data set sizes of 10 GB, 20 GB, and 40 GB respectively.




(a) (b)

**Figure 16: Throughput for query as a function of the number of back-end servers.**

Finally, Figure 17 shows where the saturation happens. We can see, from Figure 17(a) that the front-end CPU utilization increases nearly with the number of back-end servers, until it saturates (100%) at 500, 1000 and 2000 servers for data set sizes of 10 GB, 20 GB and 40 GB, respectively. Not surprisingly, the utilization of the back-end serves starts to decrease after saturation. After the front-end saturates, adding back-end server only makes the throughput and response time worse (see Figure 15 and Figure 16 above), as the front-end gets even busier handling more back-ends.



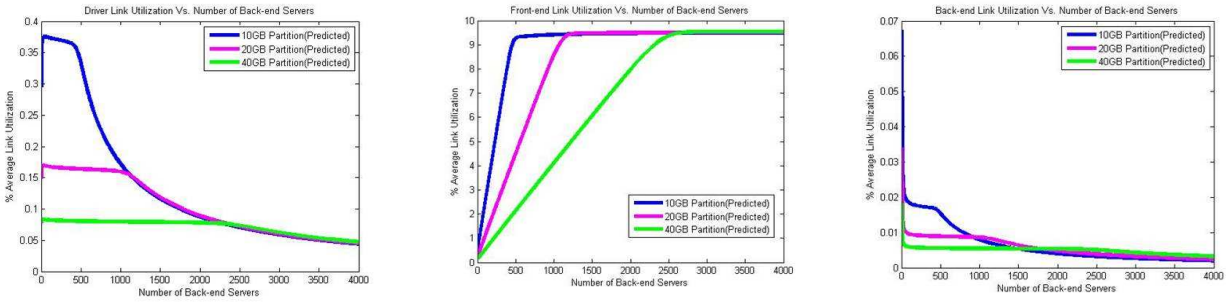

(a) (b)

**Figure 17: (a) Front-end and (b) back-end CPU utilization as function of number of servers.**

Furthermore, as we observe in Figure 18, link utilizations for the driver and back-end drop as we add servers past saturation, while link utilization for the front-end stays flat after saturation. We note that even at the front-end the utilization of the 1 Gbit/second Ethernet stays below 10%. This is not an application that stresses the communication networks, even at extreme scaling. These results also imply that we can make query scale to larger numbers of back-end servers by using a more powerful machine as the front-end. For example, even a machine 10x faster would still keep the network load within the realm of 1-Gbit/s Ethernet, and we could move to 10-Gbit/s Ethernet. Given that we would only need one front-end server for 1000+ back-end servers, the additional cost of the front-end would not be an issue.

We should note that, although our analysis predicts a saturation of query performance for a single cluster, it does not imply a natural saturation of query services. As discussed in Section 4, it is simple to extend

scalability of query by operating multiple clusters in parallel. Whether data replication or a more sophisticated solution like GPFS is used, multiple clusters provide essentially unlimited scalability for query.




(a) (b) (c)

Figure 18: (a) Driver link utilization, (b) front-end link utilization, (c) back-end link utilization

# 6 Conclusions

Search workloads behave well in a scale-out environment. The highly parallel nature of this workload, combined with a fairly predictable behavior in terms of processor, network and storage scalability, makes search a perfect candidate for scale-out. Scalability to thousands of nodes is well within reach, based on our evaluation that combines measurement data and modeling. Single clusters achieve their peak performance at around 10 back-end servers, and maintain that performance up to hundreds or even thousands of back-ends. Further scaling is possible with a multi-cluster solution.

The use of a scalable shared file system like GPFS enables more flexible configurations of search through a separation between the servers doing the search and the actual data being searched. However, the performance with GPFS is still below that with a local file system for the same number of back-ends and data set size. Based on our interaction with the developers of GPFS, we expect that those optimizations can be implemented soon, making GPFS a valuable approach to scalability of search.

# References


[1] M. Cafarella and D. Cutting. Building Nutch: open source search. ACM Queue. Vol. 2, no. 2, pp. 54-61, 2004.

[2] M. Cafarella and O. Etzioni. A search engine for natural language applications. WWW '05: Proceedings of the 14th International World Wide Web Conference. Chiba, Japan. 2005. pp. 442-452.

[3] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI'04). San Francisco, CA, December, 2004.

[4] D. M. Desai, T. M. Bradicich, D. Champion, W. G. Holland, and B. M. Kreuz. *BladeCenter system overview*. IBM Journal of Research and Development. Vol. 49, no. 6. 2005.

[5] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. *How to build a WebFountain: An architecture for very large-scale text analytics*. IBM Systems Journal. Vol. 13, no 1, 2006 pp. 64-77.

[6] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications. 2004.

[7] J. E. Hughes, P. S. Patel, I. R. Zapata, T. D. Pahel, Jr., J. P. Wong, D. M. Desai, and B. D. Herrman. *BladeCenter midplane and media interface card*. IBM Journal of Research and Development. Vol. 49, no. 6. 2005.

[8] J. E. Hughes, M. L. Scollard, R. Land, J. Parsonese, C. C. West, V. A. Stankevich, C. L. Purrington, D. Q. Hoang, G. R. Shippy, M. L. Loeb, M. W. Williams, B. A. Smith, and D. M. Desai. *BladeCenter processor blades, I/O expansion adapters, and units*. IBM Journal of Research and Development. Vol. 49, no. 6. 2005.

[9] Z. Liu, C. H. Xia, P. Momcilovic, and L. Zhang. AMBIENCE: Automatic Model Building using InfErENCE. In *Congress MSR03,* Metz, France, Oct. 2003.

[10] H. M. Mathis, J. D. McCalpin, and J. Thomas. *IBM p5 575 ultra-dense, modular cluster node for high performance computing*. IBM Systems and Technology Group. October 2005.

[11] H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, and S. R. Kunkel. *Characterization of simultaneous multithreading (SMT) efficiency in POWER5*. IBM Journal of Research and Development. Vol. 49, no. 4/5. 2005.
[12] M. Michael, J. E. Moreira, D. Shiloach, and R. Wisniewski. *Scale-up x Scale-out: A Case Study using Nutch/Lucene.* Third International Workshop on System Management Techniques, Processes, and Services (SMTPS). Held in conjunction with the 2007 International Parallel and Distributed Processing Symposium (IPDPS 2007). Long Beach, CA, March 30th, 2007.
[13] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. First Conference on File and Storage Technologies (FAST). pp. 231-244. January 2002.
[14] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. *POWER5 system microarchitecture*. IBM Journal of Research and Development. Vol. 49, no. 4/5. 2005.
[15] University of Mannheim, University of Tennessee, and NERSC/LBNL. TOP500 *Supercomputer sites*. http://www.top500.org/.
[16] E. Varki. *Mean value technique for closed fork-join networks*. In ACM Sigmetrics, pages 103-112, 1999.
[17] S. Weisberg. *Applied Linear Regression, 3rd Edition.* Wiley, 2005.
[18] L. Zhang, C. Xia, M. S. Squillante, and W. N. Mills III. *Web workload service requirement analysis: A queueing network approach*. In MASCOTS, 2002.