RC24152 (W0701-075) January 16, 2007
Computer Science

IBM Research Report

Scale-up x Scale-out:
A Case Study Using Nutch/Lucene

Maged Michael, José E. Moreira, Doron Shiloach, Robert W. Wisniewski
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

Scale-up x Scale-out: A Case Study using Nutch/Lucene

Maged Michael, José E. Moreira, Doron Shiloach, Robert W. Wisniewski

{*magedm, jmoreira, doron, bobww*}*@us.ibm.com*
IBM Thomas J. Watson Research Center
Yorktown Heights NY

Abstract

Scale-up solutions, in the form of large SMPs, have represented the mainstream of commercial computing for the past several years, with the major server vendors providing increasingly larger and more powerful machines. More recently, scale-out solutions, in the form of clusters of smaller machines, have gained increased acceptance for commercial computing. Scale-out solutions are particularly effective in high-throughput web-centric applications. In this paper, we investigate the behavior of two competing approaches to parallelism, scale-up and scale-out, in an emerging search application. Our conclusions show that a scale-out strategy can be the key to good performance even on a scale-up machine. Furthermore, scale-out solutions offer better price/performance, although at an increase in management complexity.

1 Introduction

During the last 10 years of commercial computing, 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.

We can classify the different approaches to employ multiprocessor systems for computing (both commercial and technical/scientific) into two large groups:

- *Scale-up*: The deployment of applications on large shared-memory servers.
- *Scale-out*: The deployment of applications on multiple small interconnected servers.

During the first phase of the multiprocessor revolution in commercial computing, the dominance of scale-up was clear. SMPs of increasing size, with processors of increasing clock rate, could offer ever more computing power to handle the needs of even the largest corporations. SMPs currently represent the mainstream of commercial computing. Companies like IBM, HP and Sun invest heavily in building bigger and better SMPs with each generation.

More recently, there has been an increase in interest in scale-out for commercial computing. 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 [10].)

In this paper, we study the behavior of an emerging commercial application, search of unstructured data, in two distinct systems: One is a modern scale-up system based on the POWER5 multi-core/multi-threaded processor [8],[9]. The other is a typical scale-out system based on IBM BladeCenter [3]. The systems were configured to have approximately the same list price, allowing a fair performance and price-performance comparison.

One of the more important conclusions of our work is that a “pure” scale-up approach is not very effective in using all the processors in a large SMP. In pure scale-up, we run just one instance of our application in the SMP, and that instance uses all the resources (processors) available. We were more successful in exploiting the POWER5 SMP with a “scale-out in a box” approach. In that case, multiple instances of the application run concurrently, within a single operating system. This latter approach resulted in significant gains in performance while maintaining the single system image that is one of the great advantages of large SMPs.

Another conclusion of our work is that a scale-out system can achieve about four times the performance of a similarly priced scale-up system. In the case of our application, this performance is measured in terms of queries per second. The scale-out system requires the use of multiple system images, so the gain in performance comes at a convenience and management cost. Depending on the situation, that may be worth the improvement in performance or not.

The rest of this paper is organized as follows. Section 2 describes the configuration of the scale-out and scale-up systems we used in our study. Section 3 presents the Nutch/Lucene workload that ran in our systems. Section 4 reports our experimental findings. Finally, Section 5 presents our conclusions.

2 Scale-up and scale-out systems

In the IBM product line, Systems z, p, and i are all based on SMPs of different sizes that span a wide spectrum of computational capabilities. As an example of a state-of-the art scale-up system we adopted the POWER5 p5 575 machine [7]. This 8- or 16-way system has been very attractive to customers due to its low-cost, high-performance and small form factor (2U or 3.5-inch high in a 24-inch rack). A picture of a POWER5 p5 575 is shown in Figure 1.

Figure 1: The POWER5 p5 575 SMP server.

The particular p5 575 that we used for our scale-up measurements has 16 POWER5 processors in 8 dual-core modules and 32 GiB of main memory. Each core is dual-threaded, so to the operating system the machine appears as a 32-way SMP. The processor speed is 1.5 GHz. The p5 575 connects to the outside world through two (2) Gigabit/s Ethernet interfaces. It also has its own dedicated DS4100 storage controller. (See below for a description of the DS4100.)

Scale-out systems come in many different shapes and forms, but they generally consist of multiple interconnected nodes with a self-contained operating system in each node. We chose BladeCenter as our platform for scale-out. This was a natural choice given the scale-out orientation of this platform.

The first form of scale-out systems to become popular in commercial computing was the rack-mounted cluster. The IBM BladeCenter [3],[5]solution (and similar systems from companies such as HP and Dell) represents the next step after rack-mounted clusters in scale-out systems for commercial computing. The blade servers [6] 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-processor configurations and additional memory.

Figure 2 is a high-level view of our cluster architecture. The basic building block of the cluster is a BladeCenter-H (BC-H) chassis. We couple each BC-H chassis with one DS4100 storage controller with a 2-Gbit/s Fiber Channel. 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 cluster consists of 8 chassis of blades (112 blades in total) and eight DS4100 storage subsystems

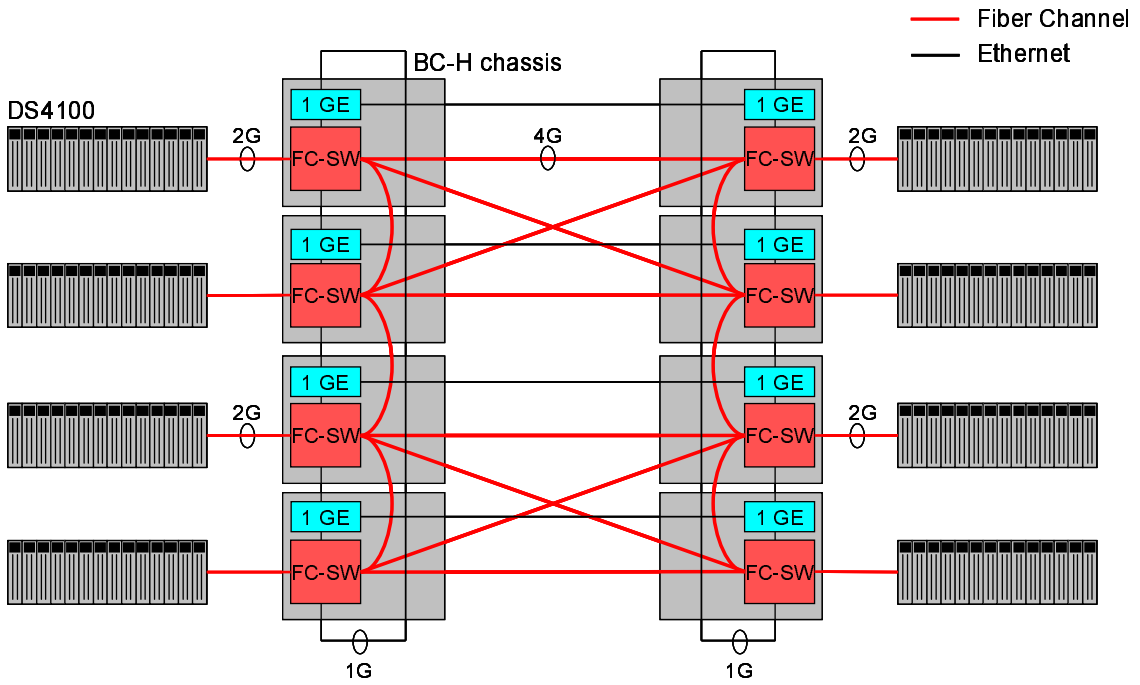


Figure 2: Hardware architecture of our BladeCenter cluster.

The BladeCenter-H chassis is the newest BladeCenter chassis from IBM. As with the previous BladeCenter-1 chassis, it 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.

Three different kinds of blades were used in our cluster: JS21 (PowerPC processors), HS21 (Intel Woodcrest processors), and LS21 (AMD Opteron processors). Each blade (JS21, HS21, or LS21) has both a local disk drive (73 GB of capacity) and a dual Fiber Channel network adapter. The Fiber Channel adapter is used to connect the blades to two Fiber Channel switches that are plugged in each chassis. Approximately half of the cluster (4 chassis worth) 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. For the experiments reported in this paper, we focus on these JS21 blades.

The DS4100 storage subsystem consists of dual storage controllers, each with a 2 Gb/s Fiber Channel interface, and space for 14 SATA drives in the main drawer. Although each DS4100 is paired with a specific BladeCenter-H chassis, any blade in the cluster can see any of the LUNs in the storage system, thanks to the Fiber Channel network we implement.

3 The Nutch/Lucene workload

Nutch/Lucene [4] is a framework for implementing search applications. It is representative of a growing class of applications that are based on search of unstructured data. We are all used to search engines like Google and Yahoo that operate on the open Internet. However, search is also an important operation within *Intranets*, the internal networks of companies. Nutch/Lucene is all implemented in Java and its code is open source. Nutch/Lucene, as a typical search framework, has three major components: (1) crawling, (2) indexing, and (3) query. In this paper, we present our results for the query component. For completeness, we briefly describe the other components.

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. The performance of crawling is usually limited by the bandwidth of the network between the system doing the crawling and the system being crawled (Internet or Intranet).

The Nutch/Lucene search framework 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 parallelism and fault-tolerance issues 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. Parallel indexing operations 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 index 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/Lucene parallel query engine is shown in Figure 3. The query engine part consists of one or more front-ends, and

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

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).

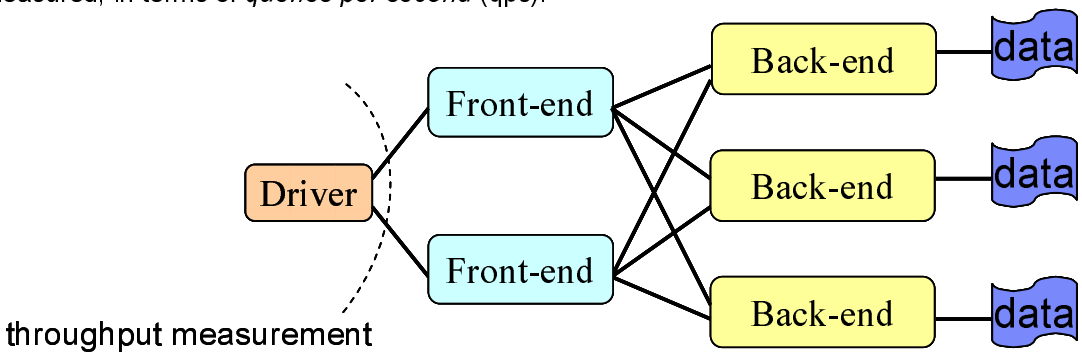


Figure 3: Overall architecture of Nutch/Lucene query.

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.

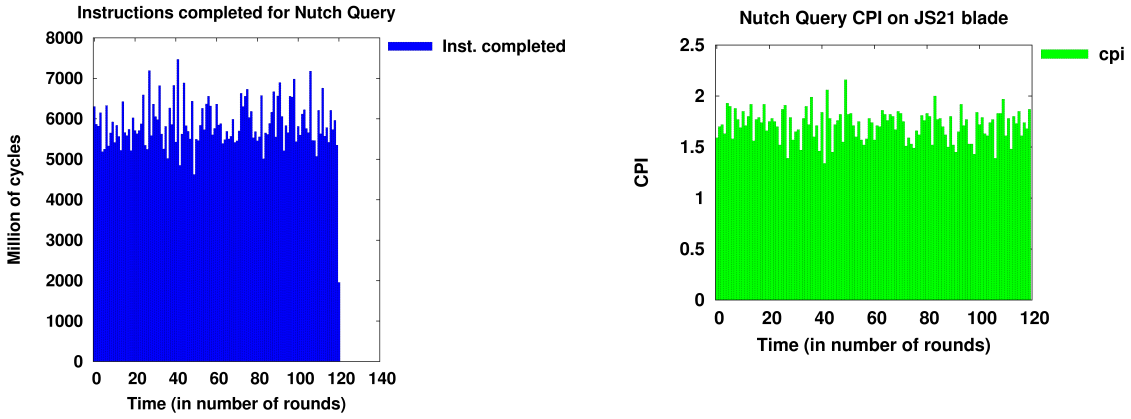
4 Experimental results

We present three different kinds of experimental results. First, we present performance counter data that allows us to characterize how applications behave at the instruction level in our JS21 PowerPC blades. We show that the behavior of the Nutch/Lucene query is similar to other standard benchmarks in the SPECcpu suite. We then report performance results from runs in our reference POWER5 p5 575 SMP machine. We compare a pure scale-up and a scale-out in a box configuration. Finally, we present scalability results from runs in the BladeCenter scale-out cluster.

4.1 Performance counter data

Using the PowerPC hardware performance counters and tools from University of Toronto [1], we performed stall breakdown analysis for the Nutch/Lucene query operation. We ran a configuration with one front-end and one back-end, each on a separate JS21 blade. The back-end operated on 10 GB of search data. We collected performance data only for the back-end, since that is the more computationally intensive component. Performance data was collected for a period of 120 seconds during steady-state operation of the back-end, and reported second-by-second.

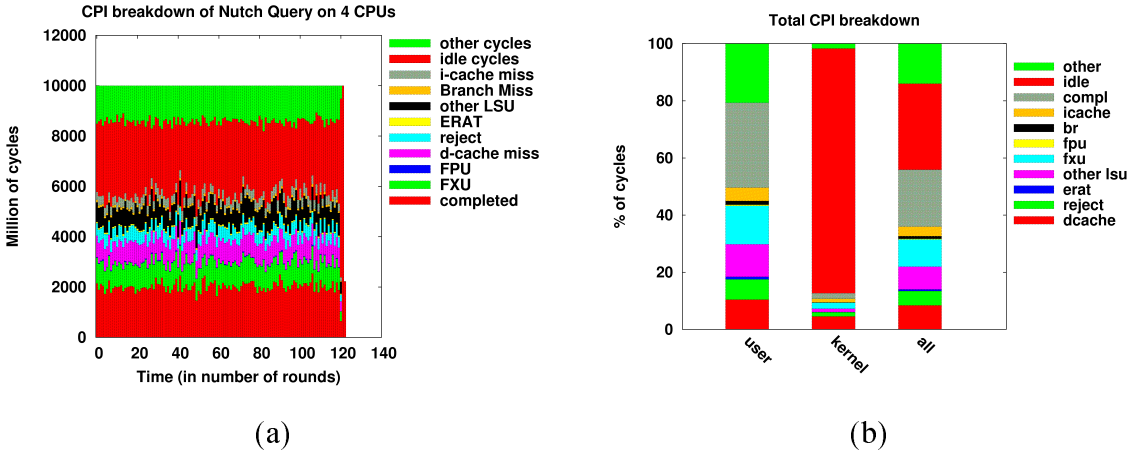
Figure 4(a) is a plot of number of instructions completed per second during the measurement interval, by all four PowerPC processors. We observe that the number of instructions completed stays mostly in the range of 5 to 7 billion instructions per second. Since there are four (4) PowerPC 970 processors running at 2.5 GHz and the PowerPC 970 can complete up to 5 instructions per cycle, the maximum completion rate of the JS21 is 50 billion instructions per second. Figure 4(b) is a plot of *clocks per instruction* (CPI) as a function of time. It is obtained by dividing, for each second, the number of cycles executed by all processors (10 billion) by the number of instructions executed during that second (from Figure 4(a)). The CPI stays in the range of 1.5-2. We note that most of the CPI values for SPECcpu 2000 in the PowerPC 970, as reported in the literature [1], are in the range 1-2. We also note that the best possible CPI for the PowerPC 970 is 0.2.



(a) (b)

Figure 4: Instruction execution profile: (a) instructions executed over time and (b) CPI over time.

From the CPI data, we can conclude that (1) the CPI for query is very far from peak, but (2) it is within what would be expected from previous experience with SPEC. More detailed information can be obtained using a stall breakdown analysis, as shown in Figure 5. Figure 5(a) is a plot of the breakdown for each 1 second period during the analysis interval. (We note that in each second there are a total of 10 billion processor cycles – 4 processors @ 2.5 GHz.) The order of the components in the plot corresponds to the order in the legend, from bottom to top. Figure 5(b) shows the average over time of the data in Figure 5(a), separated for cycles in user mode, kernel mode, and total (all).



(a) (b)

Figure 5: Stall breakdown for Nutch/Lucene query: (a) over time and (b) average.

We observe that instruction complete on only 20% of the cycles. (Multiple instructions can complete per cycle. Instructions in the PowerPC 970 complete in bundles.) From the average number of instructions executed per second (10 billion cycles/second ÷ 1.7 cycles per instruction = 5.9 billion instructions/second), we conclude that the average bundle size is approximately 3 instructions (out of a maximum of 5). Another metric we can derive is the *non-stall CPI* for query, computed dividing the number of non-stall (completed) cycles by the number of instructions. That numbers comes out at 0.34, which again is very similar to the non-stall CPI for SPECcpu [1].

Another important observation is that for a significant number of cycles (~25%), the processor is idle. That is, Linux is in its idle loop, without work to do. In principle, we should be able to reduce that idle time by increasing the load on the node. In our more extensive experiments, we found that we could keep the idle time down to between 10-15%.

Finally, we observe that cycles wasted on the I-cache or on branch mispredictions is relatively small (a few percent), stalls due to the fixed-point units account for 10% of the cycles, and stalls because of the memory hierarchy (D-cache, reject, ERAT, and other LSU) represent approximately 20% of the cycles. The fraction of cycles wasted in the memory hierarchy is similar to the fraction of cycles doing useful work. That means that a perfect memory system would only be able to at most double the performance of the existing processors. A similar benefit could be obtained by doubling the number of processors and maintaining the memory hierarchy per processor.

4.2 Experiments in a scale-up system

To run query on the p5 575, we first configured Nutch/Lucene as shown in Figure 6. We ran a single front-end and a single back-end in the machine. The actual data to be searched was stored in the external DS4100 storage controller, connected to the p5 575 through Fiber Channel. The driver was running on a separate machine, and we measured the throughput at the driver.

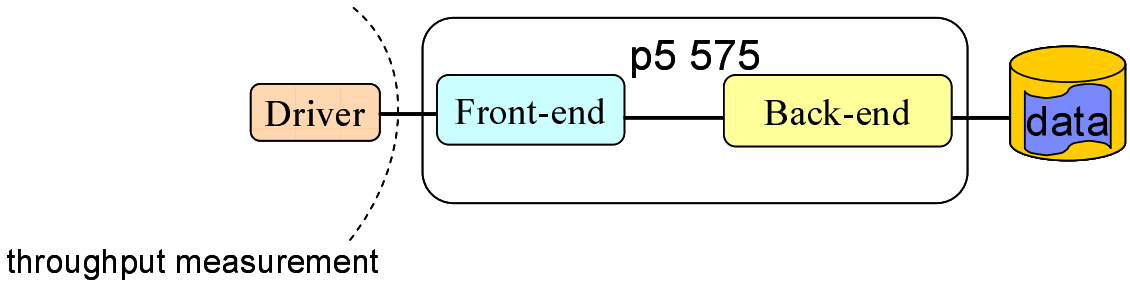


Figure 6: Running query on the p5 575: pure scale-up.

Throughput results for the configuration in Figure 6, for different data set sizes, are shown in Figure 7. We plot both the absolute queries per second (blue line, left y-axis) and the more meaningful metric of queries per second time the data set size (magenta line, right y-axis). We observe that the latter metric peaks at about 1100 queries/second*GB, for a data set size of 250 GB. We found these results disappointing, since we had measured the peak value for a JS21 blade (with only four processors) at approximately 900 queries/second*GB.

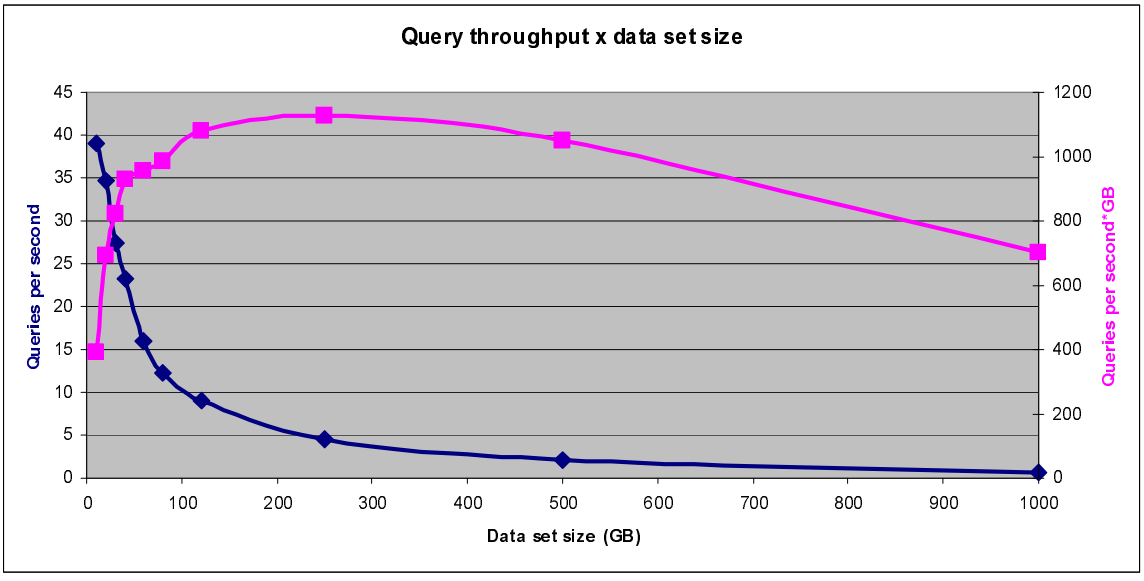


Figure 7: Throughput as a function of data set size for query on the p5 575: pure scale up.

To investigate ways to improve the performance of query on an SMP, we experimented with the configuration shown in Figure 8. In that configuration we ran multiple back-ends inside the SMP. Each back-end is responsible for 10 GB of data. So, a larger data set will use more back-ends. We call this configuration scale-out-in-a-box.

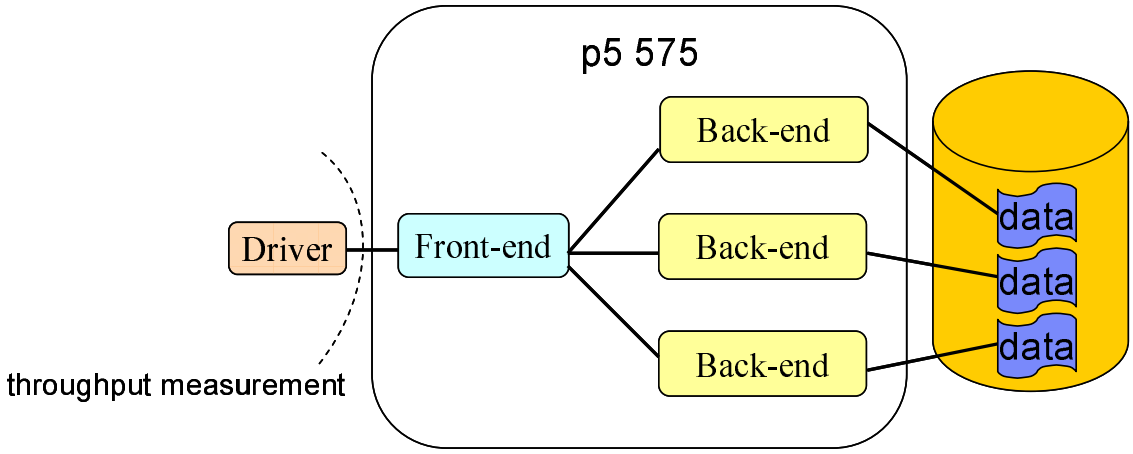


Figure 8: Running query on the POWER5 p5 575: scale-out in a box configuration.

Throughput results for the configuration in Figure 8, for different data set sizes, are shown in Figure 9. We show the results for the pure scale-up and scale-out-in-a-box in the same figure so that we can better compare them. We see a much improved behavior with this configuration. The throughput * data set size metric peaks at approximately 4000 queries/second*GB for a data set size of 250 GB.

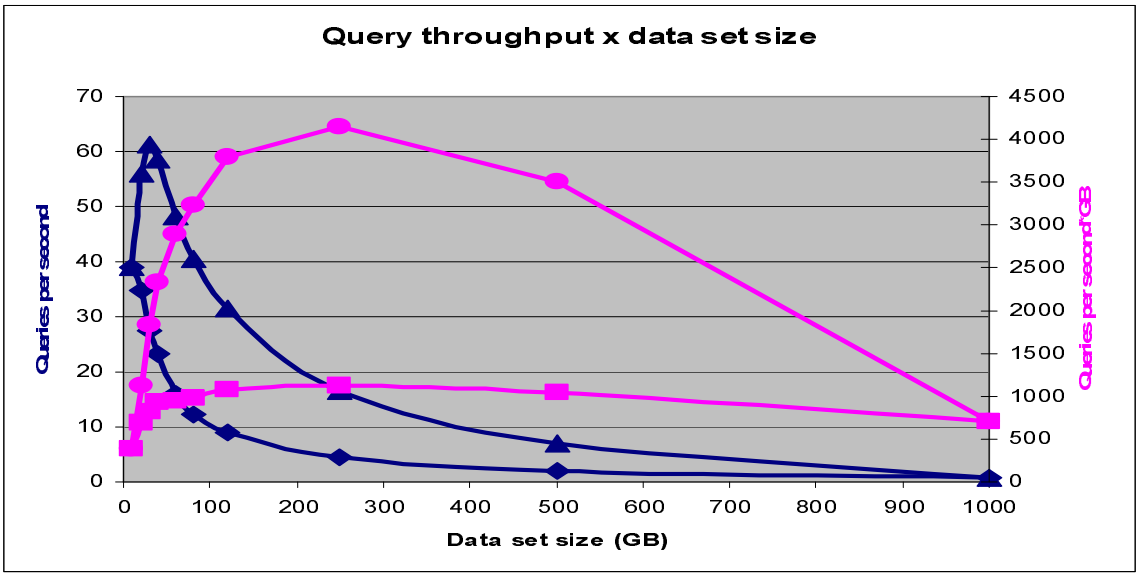


Figure 9: Throughput as a function of data set size for query on the p5 575: pure scale up and scale out in a box.

4.3 Scale-out experiments

We start by reporting results from the experiments using the configuration shown in Figure 10. 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.

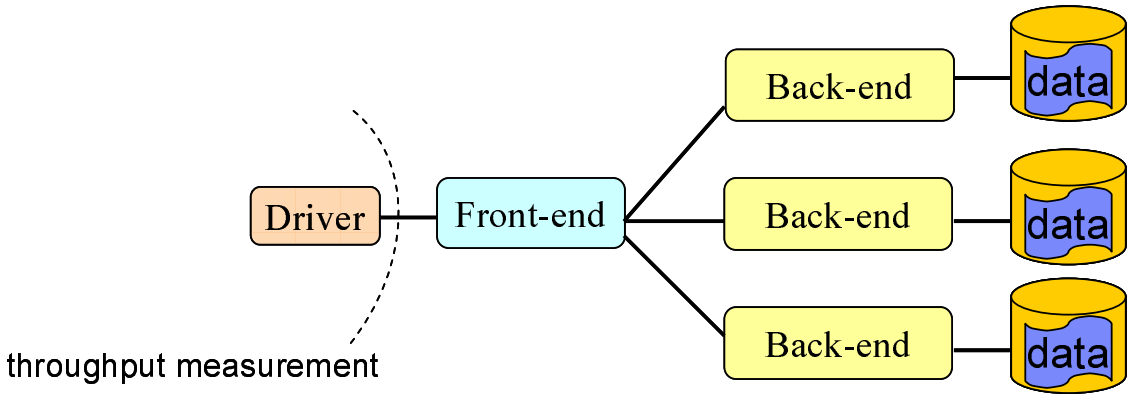
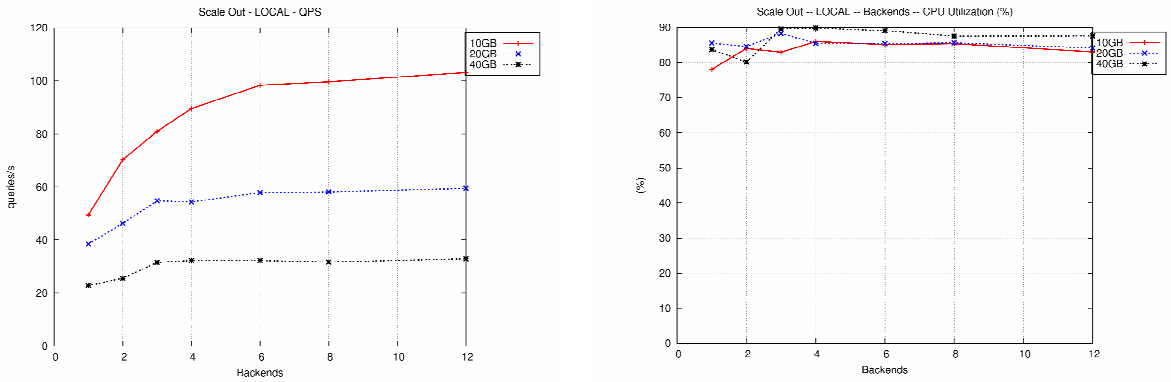


Figure 10: Configuration with each data segment in an ext3 file system in the local disk of each JS21 back-end blade.

Throughput measurements (queries per second) as a function of the number of back-ends are shown in Figure 11(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 11(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 11: 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 11(a) that the throughput 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. 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 11(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 the 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 adopt this number as the throughput that we can get from a BladeCenter chassis populated with JS21 blades. (The chassis has 14 blades, out of which we use one for front-end, 12 for back-ends, and one extra for redundancy and other services.)

We conclude this section with a comparison of performance and cost/performance between our reference scale-up system (POWER5 p5 575) and our scale-out solution. Figure 12 shows the value of the metric (throughput * data set size) for two similarly priced systems: our reference POWER5 p5 575 SMP system and one chassis of our BladeCenter cluster with JS21 blades. We see that the peak performance of the scale out solution (BladeCenter) is approximately 4 times better.

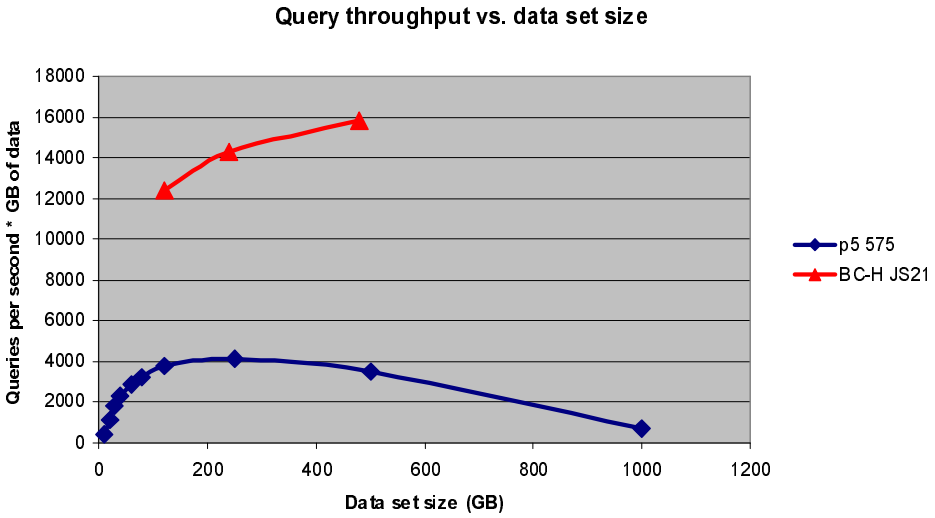


Figure 12: Comparison of two similarly priced (IBM list price) systems: a POWER5 p5 575 (scale-up) and a fully configured JS21 BladeCenter (scale-out).

5 Conclusions

The first conclusion of our work is that scale-out solutions have an indisputable performance and price/performance advantage over scale-up for search workloads. 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.

Furthermore, even within a scale-up system, it was more effective to adopt a “scale-out in a box” approach than a pure scale-up to utilize its processors efficiently. This is not too different from what has been observed in large shared-memory systems for scientific and technical computing. In those machines, it is often more effective to run an MPI (scale-out) application within the machine than relying on shared-memory (scale-up) programming.

Scale-out systems are still in a significant disadvantage with respect to scale-up when it comes to systems management. Using the traditional concept of management cost being proportional to the number of images, it is clear that a scale-out solution will have a higher management cost than a scale-up one.

References

[1] R. Azimi, M. Stumm, and R. W. Wisniewski. *Online performance analysis by statistical sampling of microprocessor performance counters*. Proceedings of the 19th annual International Conference on Supercomputing (ICS'05). Cambridge, MA, 2005.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] University of Mannheim, University of Tennessee, and NERSC/LBNL. TOP500 Supercomputer sites. http://www.top500.org/.