International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

High performance in parallel data integration: An empirical evaluation of the ratio between processing time and number of physical nodes

Caspar von Seckendorff, Eldar Sultanow

***Abstract*—**Many studies have shown that parallelization decreases efficiency [1], [2]. There are many reasons for these decrements. This paper investigates those which appear in the context of parallel data integration. Integration processes generally cannot be allocated to packages of identical size (i. e. tasks of identical complexity). The reason for this is unknown heterogeneous input data which result in variable task lengths. Process delay is defined by the slowest processing node. It leads to a detrimental effect on the total processing time. With a real world example, this study will show that while process delay does initially increase with the introduction of more nodes it ultimately decreases again after a certain point. The example will make use of the cloud computing platform Hadoop and be run inside Amazon's EC2 compute cloud. A stochastic model will be set up which can explain this effect.

***Keywords*—**Process delay, speedup, efficiency, parallel computing, data integration, E-Commerce, Amazon Elastic Compute Cloud (EC2), Hadoop, Nutch.

I. Introduction

A fairly fundamental and common requirement of Web portals is matching one's own data with data from external sources. E.g. a price comparison site has to match its product catalog with the many product catalogs from its various partners. Matching processes are however, computationally intensive, especially Fuzzy Matching, and must be run daily in order to provide up-to-date information. For many online companies, such as Producto AG[1], this matching is a core component of their business. With increasing amounts of data being imported, matching jobs increase in complexity and duration. For this reason, a scalable and high performance solution was evaluated, which is set out in this paper.

Analogous to the E-Commerce example used as a teaching case in this paper are, for instance, searching/matching addresses and personal data or searching for titles in a library catalog or archive.

Caspar von Seckendorff, Producto AG, Kreuzbergstraße 30, 10965 Berlin, Germany (e-mail: caspar.seckendorff@testberichte.de).

Eldar Sultanow, Chair of Business Information Systems and Electronic Government, University of Potsdam, August-Bebel-Straße 89, 14482 Potsdam, Germany (e-mail: eldar.sultanow@wi.uni-potsdam.de).

[1] Producto AG runs product comparison sites in Germany (www.testberichte.de), France (www.otest.fr) and in the UK (www.otest.co.uk).

A. Related Work

Parallel data processing is an issue which has been widely discussed against the backdrop of diverse applications. Data integration by Web Search Engines, especially Google, ranks as one of the most prevalent fields and has led to the development of Nutch, an Open-Source Platform for Web Search [3], [4]. Its distribution model is comprised of two layers, both of which owe a lot to contributions to information technology made by Google [5], [6]: storage (Nutch Distributed File System, NDFS) and computation (MapReduce). Typical fields of application for Nutch in web environments primarily cover Search Engines, Natural Language Processing (NLP) [7] and identification of link spam [8]. The two layers, NDFS and MapReduce, are now maintained and developed in the context of a separate project fork called Hadoop. Its operating principles, the design of cluster computing and large-scale data processing have been detailed recently in [9]. An example of its use can be seen in the distributed text index developed at HP Labs named Distributed Lucene, more information on which can be found in [10].

Scalability research concerning the inevitable trade-off between parallel execution and related communication overhead have been carried out within several scopes of reference, e.g. parallel SQL-Query executions [11], stream processor architecture [12], multi-agent implementation of cellular automata in a local network [13] and clustered processors [14]. From all of these approaches, the following points to consider become apparent:

- The total cost of execution is made up of the cost of execution on all nodes plus a number of fixed costs such as those stemming from the size of the communication overhead.
- Communication overhead may be high enough as to cancel out benefits from parallelism.
- The relationship between inter-cluster communication and cost of execution is nonlinear.

These core points are to be considered in analyzing the aforementioned trade-off and are also taken as the basis for the experiment in this paper.

It may be incidentally mentioned that this paper does not investigate queuing and prioritization problems, parallel

algorithms. The example, along with all measurements used herein is performed inside an adequately homogeneous environment. Improving MapReduce performance in heterogeneous environments is object of study in [15]. For Amazon EC2 this is particularly relevant if the load of disk and network I/O is high.

B. Problem Description

Importing and matching the products of diverse partners to one's own product data consists of parsing differently formatted files exported by partners e.g. Amazon, Shopping.com or Pangora and using an EAN (European Article Number), MPN (Manufacturer Part Number) or ASIN (Amazon Standard Identification Number) as match criterion to be compared with the products in one's own database. Matches based on these numbers are called hard-matches. However, there is also another way to match product data when such identifying information is not available; so-called soft-matching (also referred to as fuzzy matching) which involves matching products by name (and possibly other "soft" criteria such as product description or product category). In the case of Producto AG, 150,000 products are to be matched on approx. 8m products exported by partners, with only 2.5m offers from soft-matched products being displayed on its E-Commerce portal. The problem can be broken down into the following two components:

1. Masses of raw, unsorted data need to be imported into the database, but only a small amount is useful for the Website. To cache, replicate or search this data is very expensive.
2. The matching process is itself expensive and in addition, needs to be run on a daily basis.

An established approach to handling the expensive processes given in the latter component is to run them in parallel. As regards to the former; in order to decrease the amount of data stored in databases one must apply pre-filtering rules to the matching process, making the whole process more expensive and further increasing the need for process parallelisation.

The main questions followed in this paper are: What affects processing time when parallelizing a computationally intensive process? In particular: How is process delay affected by heterogeneous input data?

C. Methodology

The methodology in this paper consists of four core steps as shown in figure 1. First, a suitable architecture must be chosen and set up; this also includes servers, cluster and the distributed file system, as well as the appropriate implementation. In the second step the input and output is defined. This entails identifying factors for processing time and defining the variables (adjusting screws) for configuring the performance tests. Step three involves running performance tests to collect data for the different configurations, regulated by the previously identified variables. Finally, in the last step, the results will be analyzed and used for the construction of the model.

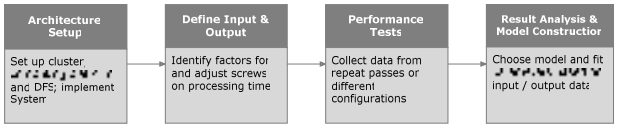

Fig. 1 This study consists of 4 steps to evaluate process delay of parallelization.

II. Architecture Setup

The process of importing and matching products and offers to be displayed on the Web pages of an E-Commerce portal can be managed by the architecture detailed in Figure 2.

Of course, products must first be manually entered; in this case, this is carried out by a dedicated editorial staff working with a CMS (Content Management System) to write these products into a database (step 1). This forms the basis of the portal's own database, which will then have to be matched with those of partners.

Following this, step two is to perform a Product-Offer-Import to import product data and offers provided by partners in various formats (e.g. XML, CSV) and store these in a generic format in Hadoop's SequenceFile. This generic format is so structured as to store data about partners' products, any associated offers and information relevant to the site's own database. This SequenceFile is mirrored to allow for load balancing, split into discrete parts which will then be distributed across the nodes. For example having five nodes and two copies means every node obtains two fifths of the total amount of data. For enabling parallelism Hadoop Framework is used, providing both HDFS (Hadoop Distributed File System, formerly NDFS) and the MapReduce programming model, which contains two fundamental steps: a map operation and a reduce operation. The principles of MapReduce are explained in [6].

Step three is the matching process itself. The process is split into a number of tasks. We parallelize the process by doing the following; each task builds an index over a part of partner's product catalog and searches in serial for the portal's products. As the set of the partner's products in each task is unique, results of the matching tasks do not need to be repartitioned across nodes in order to be merged. By default, Hadoop partitions the result of a map operation based on a hash function (e.g. hashCode mod nodeNum), therefore, were the results of the map-operation to be redistributed across all nodes, there would be no discernable benefit. In order to prevent this, a custom partitioner was implemented, which ensures that product matches will be assigned to a local node for the reduce operation. Before starting the matching process, the product table is loaded into a distributed cache and made available locally to each node in the cluster. The specific matching process relevant for the experiment is a soft-matching process. It is an inherently more computationally intensive process than hard-matching which is the primary reason for running it in parallel. The soft-matching process

uses a q-gram based algorithm for measuring string distance and similarity as detailed in [16]. This algorithm was further sophisticated by adding heuristic functions. During the soft-matching process each of our products is to be compared with each partner's products to measure the string similarity. To speed up this process, an index over the q-grams is created. The processing time increases nonlinearly when the search index increases.

Within the reduce operation (step four), matches to partner's products are merged. For this step the key is a partner's product and the value a list of products with matches found to those of the portal's own. In this step the Product-Matcher writes a new SequenceFile in the same generic format but only writes data where product matches were found. That means matching information is added and non-matched partners' products removed.

In steps five to nine the existing soft-matches are enhanced by hard-matches done manually by editorial staff from within a CMS. In the ninth step the manual matches are finally merged into the SequenceFile containing the previously found soft-matches.

In the last two steps, the offers for matched products are saved into the database to be displayed on Web pages.

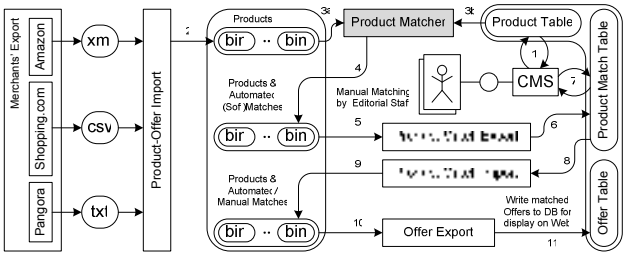

Fig. 2 The shaded element represents the most computationally intensive step within the whole architecture; the automated soft-matching.

III. Factors for Processing Time

The total duration of the process $T(N)$ depends on N, the number of physical nodes. It can be divided into two parts: $T(1)/N$, the ideal duration under which the supposition is that the process is divisible into parallelizable packages of identical size without loss of efficiency; and $T_{parallel}(N)$, the deviation from this ideal duration, which can be seen as the Process Delay. Process Delay is defined by the deviation of the slowest node from the ideal duration. Multiplying this process delay by the number of nodes will result in the overhead function as defined in [1], [2].

$$T(N) = T(1) \cdot N^{-1} + T_{parallel}(N) \quad (1)$$

For the purposes of our study, the number of nodes N is defined as the main input. This was done by changing the size of the Amazon EC2 cluster.

The model presented here assumes no other sources of overhead (e.g. coordination cost). This is a valid simplification for many map-reduce style problems. Other sources of overhead were negligible in the application described here.

TABLE I
RESULTS FOR PERFORMANCE TESTS

N	T(N)	T(1)/N	Processing time of all tasks	$T_{parallel}(N)$
1	90,775	90,775	90,775	0
2	44,778	45,387	89,289	-609
4	23,983	22,694	89,178	1289
5	19,627	18,155	89,179	1,472
10	12,140	9,077	89,669	3,063
20	6,738	4,539	88,994	2,199

The data collected contains, from left to right, the time of the worst performing node T(N), the average time, the cumulative time and the effect of parallelization (each in seconds).

The number of simultaneous tasks per single core CPU node has been set to 1 for the purpose of this study. Changing this variable would increase the number of possible configurations for the experiment and exceed the scope of this paper.

IV. Repeat-Pass Measurements on EC2

During the performance tests, the number of processing nodes is varied from one node to twenty as shown in table 1. As already mentioned, the input data is stored on the nodes (via HDFS) and each task only needs to operate on local data.

As per [6] a machine that takes an unusually long time for computation is referred to as a straggler. At this point it is to be mentioned that all nodes are of approximately similar specification. Therefore, it can be assumed that there will be no stragglers based on hardware factors alone.

Another point to be considered is the need to reboot the complete cluster after taking a measurement. This is an important step to provide a fresh, clean disk cache before the next test run is performed with a new configuration.

As previously mentioned, the processing time $T(1)$ increases when the size of the search index increases, therefore index sizes need to be kept constant. This can be done by choosing a fixed number of tasks for the complete cluster. The number of tasks for the purposes of this study will be fixed at twenty, which is equal to the maximum number of nodes. The catalog used for the study contains 841,942 partners' products whereas the number of products in the portal's database stands at 145,504. Each performance test-run starts with steps two and three (see Fig. 2), importing partners' products and loading the portal's products into distributed cache.

Table 1 shows the results for each run of the performance tests. $T(N)$ is the duration of the process, which equals the processing time of the slowest node. The third shows the ideal duration if there was no loss of efficiency. The fourth column shows the cumulative time of all tasks without idle times. $T_{parallel}$ is given in the rightmost column of the table. Time is generally measured in seconds. This data will subsequently be analyzed and used for the construction of the model.

V. Result Analysis and Model Construction

While total process duration decreases by adding nodes into the cluster, the sum of processing times of all tasks remains fairly constant.

A graphical representation of the values given in Table 1 can be found in Figure 3 where the results for time and number of nodes produce an approximately hyperbolic curve in agreement with the first term in (1). As shown in the rightmost column, process delay increases as the number of nodes is increased from 2 through 10. It is interesting to note however, that according to the data, process delay decreases as the number of nodes is increased from 10 to 20.

In order to explain this unexpected result a simple model was build based on the previously defined factors.

To discuss the result in a more general way, the processing times of individual tasks will be treated as if drawn from a stochastic distribution. Even though processing times of tasks are deterministic based on the input data, they are generally not known in advance. As can be seen in Fig. 4, task times approximately follow the shape of a normal distribution in our case.

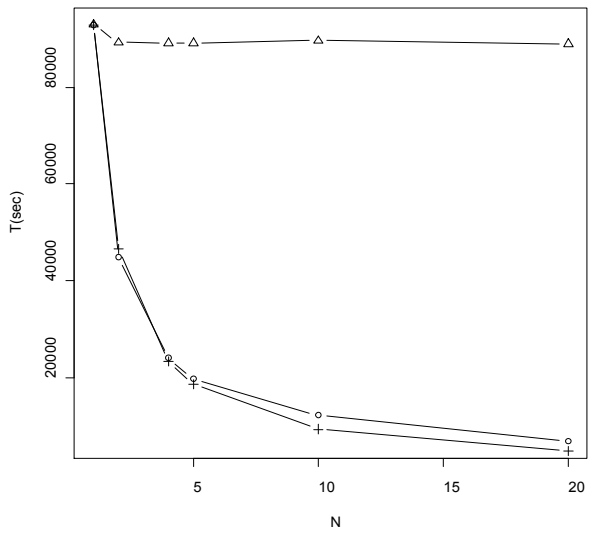

Fig. 3 Processing time of all tasks (Δ) is fairly constant. Total process duration (o) is longer than ideal process duration (+) for N > 1.

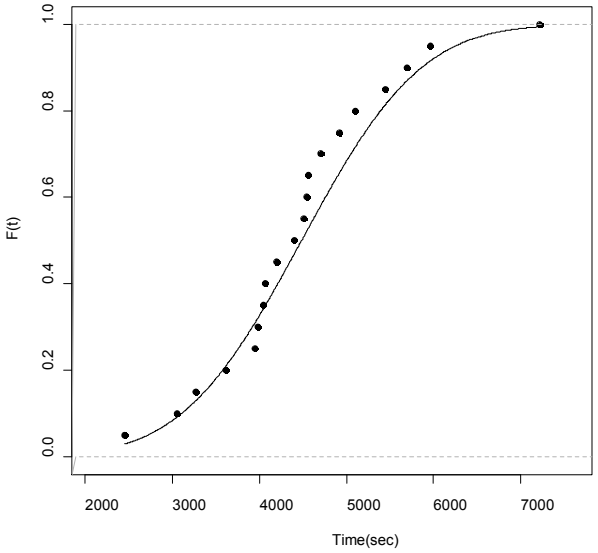

Fig. 4 The task processing times (here for N = 10) approximately follow a normal distribution.

We assume that tasks lengths t follow a normal distribution Φ with mean T_1 / k and variance σ^2. T_1 is the expected process duration free from inefficiencies, i.e. $T_1 = E(T(1))$ and k is the number of tasks ($k = 20$ for the purposes of this study). Mean and variance depend on the complexity and heterogeneity of input data.

The number of tasks on each node is $i = k / N$. We avoid the additional complexity if i is not an integer by choosing k and N appropriately (otherwise it cannot be ensured that all nodes will execute the same number of tasks).

The total process duration $T(N)$ is determined by the maximum of the nodes processing times u:

$$u = \sum_{1}^{i} t,\ u:\ \Phi\left(i \cdot \frac{T_1}{k}, i\sigma^2\right) \tag{2}$$

$$\begin{aligned} T_{parallel}(N) &= T(N) - \frac{T_1}{N} \\ &= \max\left(u - \frac{T_1}{N}\right) = \max(v),\ \ v:\ \Phi\left(0, \frac{k}{N}\sigma^2\right) \\ &= w,\ \ w \sim \Phi^N\left(0, \frac{k}{N}\sigma^2\right) \end{aligned} \tag{3}$$

The expected value of $T_{parallel}(N)$ was estimated by simulating 100.000 samples, based on (3) with σ^2 taken from the data of our previous measurements:

$E(T_{parallel}(1)) \approx -2,4$
$E(T_{parallel}(2)) \approx 1884,3$
$E(T_{parallel}(4)) \approx 2428,9$
$E(T_{parallel}(5)) \approx 2454,5$
$E(T_{parallel}(10)) \approx 2300,1$
$E(T_{parallel}(20)) \approx 1972,4$

This shows that from $N = 1$ to $N = 5$ the expected process delay increases, whereas from $N = 5$ to $N = 20$ it decreases. Figure 5 displays the cumulative distribution function of the term $T_{parallel}$ for $N = 1$, 5 and 20.

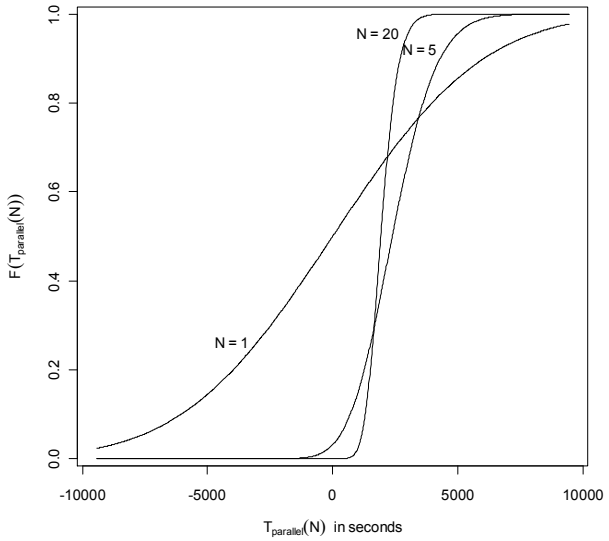

Fig. 5 The mean of $T_{parallel}$ increases from $N = 1$ to $N = 5$ and decrease from $N = 5$ to $N = 20$.

VI. Conclusion and Future Work

Our results show that one source of overhead in parallelization is non constant task lengths. In practice it is often difficult to partition the input data in advance into parts of equal computational complexity. Therefore non constant task lengths will in fact present a problem: It would be desirable to increase the number of tasks each node is processing sequentially and simultaneously decrease the duration of each single task. Better balancing of workload could be achieved by dynamically reducing the number of tasks for "slower" nodes during the process. Decreasing task length is however not always possible, at least not without introducing other inefficiencies. In our case the overhead of building up an index before the start of the task would be increased with each task and the index size would become inoptimal. In other scenarios additional communication costs between nodes (e.g. if input data is not stored locally) could also play a role.

Future studies could investigate the effect on non-constant task lengths in parallel data integration in the context of other sources of overhead. To reduce the problem in Hadoop, one could define an algorithm for splitting input data more intelligently. This algorithm would not only consider raw data size. It should sample the data and take its complexity into account, which is application specific.

References

[1] Kumar, V.: Introduction to Parallel Computing, 2nd edition. Addison-Wesley Longman Publishing Co., Inc., (2002)
[2] Eager, D. L., Zahorjan, J., Lozowska, E. D.: Speedup Versus Efficiency in Parallel Systems. IEEE Transactions on Computers, Vol. 38, No. 3, pp. 408--423 (1989)
[3] Cutting, D.: Nutch: an Open-Source Platform for Web Search. In: Beigbeder, M., Yee, W. G. (eds.) Workshop on Open Source Web Information Retrieval (OSWIR), pp. 31--33 (2005)
[4] Khare, R., Cutting, D., Sitaker, K., Rifkin, A.: Nutch: A Flexible and Scalable Open-Source Web Search Engine (2004)
[5] Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google File System. SOSP '03: Proceedings of 19th ACM symposium on Operating systems principles, ACM Press, pp. 29--43, NY (2003)
[6] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004)
[7] Cafarella, M. J., Etzioni, O.: A Search Engine for Natural Language Applications. WWW '05: Proceedings of the 14th international conference on World Wide Web, pp. 442--452. ACM Press, NY (2005)
[8] Drost, I., Scheffer, T.: Thwarting the Nigritude Ultramarine: Learning to Identify Link Spam. ECML, pp. 96--107 (2005)
[9] Kimball, A., Michels-Slettvet, S., Bisciglia, C.: Cluster Computing for Web-Scale Data Processing. SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education, ACM, pp. 116--120 (2008)
[10] Butler, M. H., Rutherford, J.: Distributed Lucene : A distributed free text index for Hadoop. HP Laboratories (2008)
[11] Hasan, W., Motwani, R.: Optimization Algorithms for Exploiting the Parallelism-Communication Tradeoff in Pipelined Parallelism. VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., pp. 36--47 (1994)
[12] Ahn, J. H., Erez, M., Dally, W. J.: Tradeoff between Data-, Instruction-, and Thread-Level Parallelism in Stream Processors. ICS '07: Proceedings of the 21st annual international conference on Supercomputing, ACM, pp. 126--137 (2007)
[13] Amini, H., Kazakov D., Ridge, E.: Parallelism vs Communication Overhead Trade-off in a JADE Multi-Agent Implementation of Cellular Automata. The First International Symposium on Nature-Inspired Systems for Parallel, Asynchronous and Decentralised Environments (NISPADE), AISB convention, Bristol (2006)
[14] Balasubramonian, R., Dwarkadas, S., Albonesi, D. H.: Dynamically managing the communication-parallelism trade-off in future clustered processors. SIGARCH Comput. Archit. News, ACM, pp. 275--287 (2003)
[15] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., Stoica, I.: Improving MapReduce Performance in Heterogeneous Environments. 8th Symposium on Operating Systems Design and Implementation, pp. 29--42 (2008)
[16] Ukkonen, E.: Approximate string-matching with q-grams and maximal matches. Theor. Comput. Sci., Elsevier Science Publishers Ltd., 92, pp. 191--211 (1992)