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.

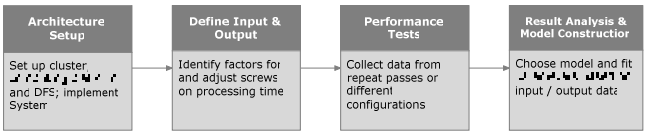

Fig. 1 This study consists of 4 steps to evaluate process delay of parallelization.

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