Structuring All-Pairs as a MapReduce Application

Cyril Briquet, Stéfan Sinclair and the With Criminal Intent Research Group
CSMM Department, McMaster University, Hamilton, ON
cyril.briquet@acm.org, sgs@mcmaster.ca

***Abstract*—All-Pairs consists of applying a computation to all pairs of data pieces from a couple of data sets. Computing All-Pairs at scale, despite its apparent simplicity, is challenging, mainly because of the massive amounts of data to handle. In this poster paper, we describe how to structure All-Pairs as a MapReduce application, which permits arbitrarily high task parallelism. This involves separating data designation from data transfer, transferring and handling input data efficiently by relying on a distributed data transfer mechanism complementarily to the MapReduce model, and guaranteeing a total order on the generated output data. An interesting insight is that one of two main touted strengths of MapReduce - its ability to efficiently handle large input data - is not the key factor for All-Pairs. However, the other main strength, i.e. the ability of individual compute nodes to handle large amounts of data using an external memory mechanism, of course is absolutely required to operate at scale. We illustrate our proposed structuring with an application scenario consisting of estimating document similarity in an historical collection of judicial records.**

***Keywords*-All-Pairs; Cloud; Cluster; MapReduce;**

I. Introduction

The All-Pairs computing abstraction [1] consists of applying a given computation to all pairs of data pieces that can be generated from a couple of data sets. All-Pairs is relevant to many application domains including bioinformatics, data mining [2] and, as the application scenario illustrating this poster paper, estimation of document similarity [3] in an historical collection of judicial records [4].

Despite its apparent simplicity, computing one large instance of the All-Pairs problem in an undedicated environment, such as a general-purpose compute cluster, is challenging. Indeed, if not handled suitably, transfer of input data and output data constitute performance bottlenecks, both at the node-level and the cluster-level.

In this poster paper, we describe how to structure All-Pairs as a MapReduce [5] application. Relying on the MapReduce model enables to leverage the reliability and scaling guarantees it offers. Namely: the reexecution mechanism guarantees eventual task completion; the out-of-core [6] merge mechanism guarantees that no individual compute node is overloaded by large amounts of data; and the scheduling mechanism enables to leverage large amounts of compute nodes.

Our proposed structuring of All-Pairs as a MapReduce application is comprised of the following: firstly, separating data designation from data transfer; secondly, transferring and handling input data efficiently; thirdly, guaranteeing a total order on the generated output data. MapReduce is often lauded for its ability to efficiently handle large input data. Interestingly, one insight derived from our research project is that computing All-Pairs at scale does not necessarily require this ability. Indeed, by separating data designation from data transfer, the input data, i.e. the key/value pairs of the so-called *map phase*, remain modest in size while most of the data is transferred as side data, i.e. so-called *dictionary* data. Distributed data transfer mechanisms that can be used to transfer this side data, complementarily to the MapReduce model, include BitTorrent [7], HDFS [8], Lustre [9].

The rest of this poster paper is structured as follows: Section II defines the All-Pairs problem, Section III provides a brief summary of the MapReduce model, Section IV introduces our application scenario, and Section V describes how we propose to structure All-Pairs as a MapReduce application, given our application scenario.

II. The All-Pairs Problem

The All-Pairs computing abstraction [1] basically consists of applying a given computation, say function F, to all pairs of data pieces from a couple of data sets, say A and B, in order to yield a result matrix, say M [2]:

All-Pairs(set A, set B, function F) returns matrix M:
Compare all elements of set A to all elements of set B via function F, yielding matrix M, such that M[i,j] = F(A[i],B[j])

Despite its apparent simplicity, computing All-Pairs at scale is challenging [2]. One challenge is the overload of individual compute nodes; it can be prevented by relying on an external memory mechanism [6]. Other challenges reside in making available almost all of the input data to each compute node, and the lack of scalability of centralized data transfers; both are typically addressed [2] by limiting task parallelism.

III. MapReduce Model

MapReduce [5] is a programming model that facilitates automatically running distributed applications on clusters of commodity computers. *"Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key* [5]. There are thus two phases in the MapReduce model: the so-called *map phase* and *reduce phase*.

A middleware based on the MapReduce model transparently partitions and transfers input data, schedules the execution of multiple instances of the map and reduce functions, transfers intermediate data and output data, and handles failures of compute nodes by rescheduling computations. Input or intermediate data subsets too large to fit within the memory of an individual compute node are handled by relying on an external memory mechanism [6].

IV. Application Scenario

Our application scenario consists of estimating document similarity [3] in an historical collection of judicial records, the Old Bailey corpus [4]. The corpus comprises a set of 200,000 XML documents. Each record is the lemmatized text extracted from a TEI-encoded XML document. The objective is to solve the All-Pairs problem for the Old Bailey corpus, evaluating each of the 40 billion pairs of records (related MapReduce works are designed to prune as many evaluations as possible [10]) using the Normalized Compressed Distance (NCD) [11]. The whole corpus of 200,000 TEI-encoded XML documents is 1.7 GB in size and the set of lemmatized textual records is only one tenth of that. There is thus little raw input data to transfer comparatively to the problem size.

The All-Pairs problem is offered as a tool of the Voyeur Tools [3] cloud-based text analytics platform. Voyeur Tools processes bursts of user requests on undedicated HPC clusters. The compute environment of Voyeur Tools is an HPC cluster based on 8-core CPUs with access to a high performance distributed file system (Lustre [9]). The All-Pairs tool is expected to be publicly released by the end of 2010.

Voyeur Tools relies on Apache Hadoop [8], a popular open source middleware based on the MapReduce model. Typical Hadoop deployments rely on the HDFS distributed file system to transfer and store input data and gather output data. Hadoop compute nodes can also access additional file systems, whether local or distributed, if available.

V. All-Pairs as a MapReduce Application

A. Data designation vs. data transfer

Based on our application scenario, the input data is comprised of one set of textual records, each associated with a unique identifier (r-id, for short). A record and its r-id weigh 4 kB on average. The total size of the 40 billion pairs of records would be 320 terabytes without any optimization. We propose to abstract data designation from data transfer. The textual contents of all records are gathered into one file, referred to as the so-called *dictionary* data. An index, using r-ids as keys and byte offsets as values, enables to locate textual records. By adding one level of indirection, the size of the input data transferred as a *dictionary* file is significantly reduced ($\sim$800 MB, that is 0.004% of the size of raw data).

Pairs of r-ids transferred as input data are grouped into several files, the so-called *input splits*, with enough r-ids per split to keep the overhead low while ensuring high task parallelism and low fault-recovery cost.

B. Input data transfer and handling

The pairs of r-ids are transferred by the MapReduce middleware. The *dictionary* (see above) must be replicated to compute nodes, with HDFS [8] (typically 3 replicas, more as needed) or Lustre [9] (data blocks are replicated on demand). BitTorrent [7] could also be used.

To compute the NCD of a pair of records (see above), the compressed sizes of both individual records of a pair, as well as their concatenation, are required. Each compute node maintains an in-memory hierarchical cache of record textual contents and their compressed sizes, in order to minimize the computations and file system reads.

C. Total order on output data

The computed output data resulting from the evaluation of record pairs are gathered and sorted into output shards (the so-called *reducers*) by the MapReduce middleware. Each record pair is preassigned (before running the MapReduce application) to a specific output shard. To maintain a total order of record pairs over all output shards, the output of a record pair is comprised of the preassigned output shard, the leftwise r-id and the computed NCD value.

D. Specific Limitations and Future Work

The implementation of our proposed structuring of All-Pairs as a MapReduce application could be exposed as an API so that it can be considered as a true computing abstraction [2]. Additional experimental results will be required to confirm the promising early results and help tune the design parameters. Early results show that computing All-Pairs for 20% of the records currently requires 40 minutes: $\sim$20 min. to generate the dictionary data, $\sim$20 min. to run the MapReduce application using 28 nodes (8-core CPUs) with 2 map slots per core and 2 reduce slots per node.

Acknowledgments

This work was made possible by Digging into Data (www.diggingintodata.org) and by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET: www.sharcnet.ca) and Compute/Calcul Canada.

References

[1] C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn, "All-Pairs: An Abstraction for Data-Intensive Computing," in *Poster Proc. Grid Computing conference*, 2007.

[2] ——, "All-Pairs: An Abstraction for Data-Intensive Cloud Computing," in *Proc. IPDPS*, Miami, FL, 2008.

[3] "Voyeur Tools." [Online]. Available: http://voyeurtools.org/

[4] "The Proceedings of the Old Bailey, 1674-1913." [Online]. Available: http://www.oldbaileyonline.org/

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. OSDI*, San Francisco, CA, USA, 2004.

[6] J. S. Vitter, "External Memory Algorithms and Data Structures: Dealing with Massive Data," in *ACM Computing Surveys*, 2001, vol. 33, no. 2.

[7] C. Briquet, X. Dalem, S. Jodogne, and P.-A. de Marneffe, "Scheduling Data-Intensive Bags of Tasks in P2P Grids with BitTorrent-enabled Data Distribution," in *Proc. UPGRADE-CN'07, HPDC Workshops*, Monterey Bay, CA, USA, June 2007.

[8] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2009.

[9] "Lustre." [Online]. Available: http://www.lustre.org/

[10] G. De Francisci Morales, C. Lucchese, and R. Baraglia, "Scaling Out All pairs Similarity Search with MapReduce," in *Proc. SIGIR*, Geneva, Switzerland, July 2010.

[11] R. Cilibrasi and P. Vitnyi, "Clustering by Compression," in *IEEE Trans. Information Theory*, 2005, vol. 51, no. 4.