

Operator and Workflow Optimization for High-Performance Analytics

Hans Vandierendonck, Karen L. Murphy, Mahwish Arif,
Jiawen Sun and Dimitrios S. Nikolopoulos*

Queen's University Belfast
Belfast, United Kingdom

{h.vandierendonck,k.l.murphy,m.arif,jsun03,d.nikolopoulos}@qub.ac.uk

ABSTRACT

We make a case for studying the impact of intra-node parallelism on the performance of data analytics. We identify four performance optimizations that are enabled by an increasing number of processing cores on a chip. We discuss the performance impact of these optimizations on two analytics operators and we identify how these optimizations affect each another.

Keywords

Data analytics; high-performance analytics; intra-node parallelism

1. INTRODUCTION

In the era of *big data*, data analytics represent an increasing fraction of the processing cycles spent in data centres. While in classical data management and analytics, operators could be described within the bounds of the SQL domain-specific language, this is no longer the case with data analytics for big data. In big data, the operators are diverse, as is the data they are operating on, and can involve any algorithm to transform, classify or structure the data at hand. As the structure of the data and the associated operators are ill-defined, so is the scope of data analytics operators.

In order to achieve low processing times, operators require careful design and must be highly optimized. Where efficient SQL statements may, to a large extent, be written by domain experts and automatically optimized, this is definitely not the case for analytics operators. Moreover, optimization of the operators is often counter-intuitive as it involves sparse data sets, which are less deeply covered in text books and as

*This work is supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under the ASAP project, grant agreement no. 619706, and by the United Kingdom EPSRC under grant agreement EP/L027402/1.

such less familiar to developers. Pitfalls exist that tend to make algorithms on sparse data sets memory-bound where they could be more efficient and compute-bound. As such, libraries with high-performance implementations of common operators are provided [3, 6, 13].

While individual operators are commonly provided by various libraries, a useful computation typically requires a workflow that links together multiple operators. Very often, these operators communicate through data stored on disk, which induces redundant operations, such as I/O, involving disk access, parsing and data conversions. An alternative solution is to create single binaries that encapsulate a complex workflow [14]. Such a solution has the potential of significantly higher processing rates by avoiding unnecessary I/O.

This paper presents techniques for optimizing operators and workflows in order to achieve high-performance analytics. Within this area, we identify and characterize four widely applicable optimisations. 1. Assuming (highly) parallel nodes (Moore's Law predicts that all future nodes will be increasingly more parallel), we demonstrate that utilizing parallelism within operators is extremely important to improve performance. The motivation behind using parallelism are the observations that (i) parallelism allows to hide I/O latencies, and (ii) many analytics problems are compute-bound rather than I/O-bound, an observation that goes against intuition [2, 11].

2. Input and output contribute to a large proportion of the execution time due to the size of data sets and the potentially low amount of computation performed per byte transferred. I/O operations benefit, however, also from intra-node parallelism, allowing on the one hand to read independent files concurrently, and on the other hand overlapping data processing with disk and network access latency.

3. While big data frameworks often steer towards dumping intermediate data sets to disk, the overhead of I/O and storing intermediate data sets are significant. This overhead can be avoided by fusing operators in workflows into single executable images and by feeding the data from one operator to the next.

4. The choice of internal data structures used in analytics operators is determining for the performance of the operator. We demonstrate a 3.4 fold speedup by interchanging one standardized data structure for another. However, this result depends also on the degree of intra-node parallelism utilized. As such the optimization problem is non-trivial.

The goal of this paper is to demonstrate the importance of these considerations for implementing analytics queries.

Table 1: Data set description.

Input	Documents	Bytes	Distinct words
Mix	23432	62.8 MB	184743
NSF Abstracts	101483	310.9 MB	267914

The remainder of this paper is structured as follows. In Section 2 we discuss our system assumptions and software environment. In Section 3 we analyse two typical operators, one text processing and one numeric operator, and characterize the impact achievable by the identified performance optimizations. In Section 4 we discuss related work.

2. SETUP

We start our investigation at a small scale, focusing on the activities on a single node as these allow us to better understand the performance of operators and workflows.

Performing analytics on a single node is important as a single-node can be built with a large amount of working memory (up to 16 TB) and many processing cores (over a 100). Such a system could efficiently process many real-world data sets. However, we expect that our conclusions remain valid when applied to scale-out systems, as optimizing the performance of nodes in isolation is crucial to optimize the system overall.

To test the importance of the identified optimisations, we implement two analytics operators in the Cilkplus extension of C++, a programming language designed for high-performance and parallel computing at MIT, first developed over two decades ago and continuously refined since then. Cilkplus, now commercialized by Intel, supports the construction of parallel tasks through language constructs that express parallelism and vectorization (SIMDization) in an easily accessible way. In the Cilkplus model, each thread of computation is bound to a processing core. The principles utilized should apply to other languages and parallel constructs, e.g., Java streams.

We study two operators: term frequency-inverse document frequency (TF/IDF) and K-means clustering. TF/IDF extracts words from text documents and rates the importance of a word on the basis of its frequency of occurrence within a specific document as well as within the whole set of documents. K-means clustering is an unsupervised classification technique that allows for the grouping of similar data items described as numeric vectors.

3. ANALYSIS

3.1 Intra-Node Parallelism

Many problems in data mining are trivially compute-bound, especially learning algorithms using neural networks, support vector machines and the like, which utilize computationally demanding hyperbolic functions and can require many iterations to train the model. It should go without saying that algorithms like these can be accelerated using high degrees of intra-node parallelism.

K-means clustering is perhaps one of the cheapest unsupervised learning algorithms. As such, we will use K-means clustering to demonstrate that data analytics operations benefit from intra-node parallelism. Figure 1 shows the self-relative speedup of the K-means clustering algorithm on

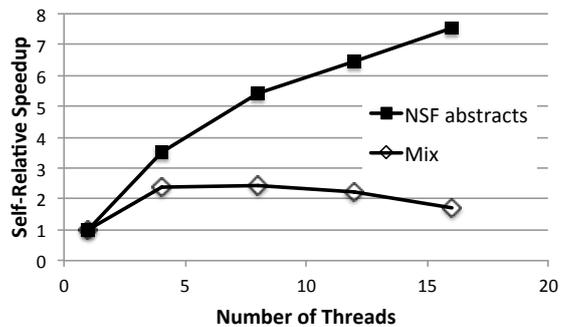


Figure 1: Self-relative performance scalability of the K-Means operator.

our two datasets (Table 1). We use the algorithm to assign documents to one of 8 clusters based on their normalized TF/IDF scores.

The self-relative speedup shows how much performance is improved by utilizing multiple CPU cores. The speedup obtained is sensitive to the data set operated on: The *NSF Abstracts* data set has about 100,000 documents and is sped up nearly 8 times using intra-node parallelism. The *Mix* data set has around 23,000 documents, which is sufficient only for a 2.5 x speedup. This effect is easily explained by the parallel loops in K-means clustering, which are all loops iterating over the documents. As the number of documents grows, so does the parallel scalability.

The execution time of our implementation is furthermore short in comparison to other implementations. We compared the execution time of our K-means clustering implementation against WEKA [3] (version 3.6.13). Using the “SimpleKMeans” algorithm, a single-threaded K-Means algorithm, on the same data sets requires over 2 hours, after which we aborted the execution. In contrast, executing our implementation sequentially required 3.3s and 40.9s for the *Mix* and *NSF Abstracts* data sets respectively. Note that while we did not see the execution of WEKA through to the end, we have verified that our WEKA installation works correctly on small data sets.

While our implementation is significantly faster than WEKA, this is not automatic. Several key optimisations were required to achieve the performance of our algorithm: (i) Using sparse vectors to represent inherently sparse data. (ii) Recycling data structures throughout the K-means iterations to avoid redundant data copies and memory pressure. E.g., we do not create new objects during the iterations of the K-means algorithm.

The conclusion of this experiment is thus that (i) intra-node parallelism is an important opportunity to accelerate data analytics, especially on larger data sets; (ii) the implementation and the choice of data structures has a huge influence on execution time; (iii) parallelism can be exploited without casting the algorithms in map/reduce form.

3.2 Parallel Input

A code that is well-optimized and where CPU is a bottleneck can also benefit from parallelizing I/O operations. Under these circumstances, CPU utilization is high and I/O resources are underutilized, including local disk and network resources. Intra-node parallelism can thus increase the utilization of disk and network resources.

In this section we study the problem of calculating the

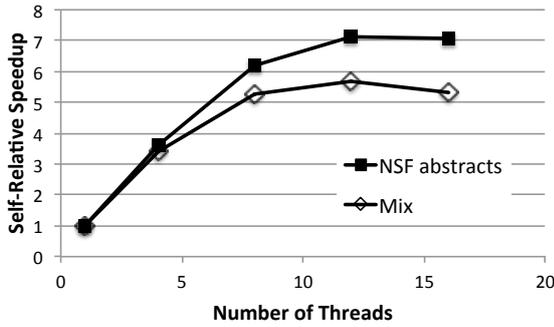


Figure 2: Self-relative parallel scalability of the TF/IDF operator.

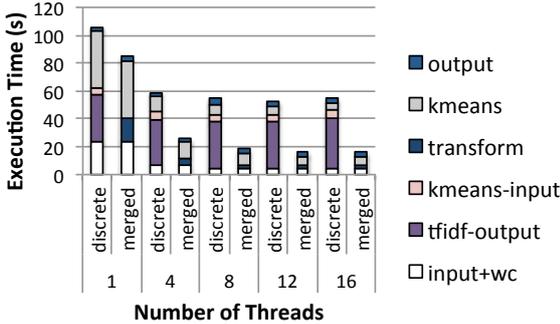


Figure 3: Execution time of the TF/IDF-K-Means workflow when executing the TF/IDF and K-Means operators as discrete steps communicating through file I/O, versus a merged operator with storage of the TF/IDF scores. Uses the *NSF Abstracts* input.

term frequency-inverse document frequency (TF/IDF) [10] property of a set of documents. Our implementation collects term frequencies (word counts) for each of the documents in the set. Moreover, a list of all unique terms across the documents is constructed. This list is annotated with the number of documents where the word occurs. In a first phase, the per-document term frequencies and the overall term-document count properties are collected using dedicated hash tables, mapping a word to a term frequency or an overall document count. In a second phase, we calculate for each document the per-term TF/IDF score using the hash tables described above. For each document, a sparse TF/IDF vector is constructed, sorted by term IDs and written to the output file in Attribute-Relation File Format (ARFF) format [3]. The first phase can be executed in parallel for each of the documents. The main limitation to obtain speedup here is bandwidth to the storage system. The second phase is not parallelized as the ARFF format does not facilitate parallel output.

While the TF/IDF problem is mainly concerned with data input, tokenization and hash table operations, it benefits strongly from intra-node parallelism (Figure 2). It speeds up by nearly 6-fold for the *Mix* data set and by 7-fold for the *NSF Abstracts* data set. Parallelizing output is important as well. However, file formats are often designed in such a way that parallel I/O becomes hard.

3.3 Workflow Fusion

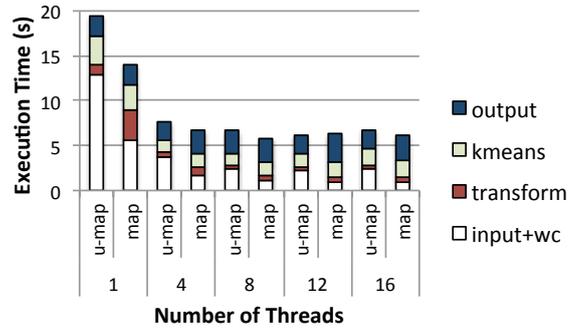


Figure 4: Execution time of the TF/IDF-K-Means workflow on the *Mix* input using a `std::unordered_map` (u-map) or a `std::map`.

As pointed out above, I/O is both costly and hard to parallelize. As such, avoiding I/O is always a good optimization. Figure 3 shows the execution time of the TF/IDF-K-Means workflow when executing the TF/IDF and K-Means operators as discrete operators that communicate by storing the intermediate TF/IDF scores on disk, versus a merged operator without storage of the intermediates. The results clearly demonstrate that dumping data to disk has a high latency. In this experiment, the data is dumped to a local hard disk. Both the output of the TF/IDF scores and the subsequent input are executed by a single thread because the file format utilized (ARFF [3]) does not easily support parallel I/O. In contrast, transforming the data when it is stored in-memory is much faster and parallelizes well.

The presence of intra-node parallelism is an important differentiator as to whether I/O bears much overhead or not. On a single-threaded execution, I/O increases execution time by 36.9%. On 16 threads, however, I/O makes the execution 3.84 times slower because it does not parallelize.

3.4 Data Structures

Algorithms use data structures to store input, output and internal data sets, The choice of these data structures impact performance. In the case of TF/IDF, the key data structures are the dictionaries storing unique words and their frequencies. Figure 4 shows the execution time of TF/IDF-K-Means workflow on the *Mix* data set and a varying number of threads. Results for the target *NSF Abstracts* data set are more dramatic.

The results demonstrate that the input and word-count step (“input+wc” in Figure 4) is faster when using the `std::map` data structure as opposed to the `std::unordered_map` data structure. The first is implemented as a red-black tree, while the latter is implemented as a hash table. Moreover, the unordered map is pre-sized to hold 4K items to minimize resizing overhead.

While reading documents and counting words is faster with a map, the subsequent data transformation step is slower using a map, especially on one thread. This follows as the input and word-count phase is write-intensive, consisting of frequent insertion of values in the dictionary. Insertion in the unordered map (a hash table) is inefficient due to (i) resize operations, which requires re-hashing all elements, (ii) memory pressure, as the array underlying the hash table is by construction both sparse (to approximate $O(1)$ opera-

tions) and very large (due to the data sets used). In contrast, the transformation step performs only lookups on the hash table, which are known to be faster on the unordered map $O(1)$ as opposed to the map $O(\log n)$.

However, the transformation step scales much better with an increasing number of threads when using the map: it scales to 6.1x on 16 threads using the map, while it scales only to 3.4x using the unordered map data structure. This is in part due to the memory consumption. In particular, using the *Mix* data set, main memory consumption is 420MB with the map, while it rises to 12.8GB using the unordered map.

Likewise, the output phase performs lookups only on the dictionaries and thus favours the unordered map. Moreover, the output phase is hard to parallelize.

We conclude that selection of the internal data structures has a significant impact on execution time. Moreover, different steps of a workflow may execute faster using different data structures. As such, the choice of internal data structure must be taken judiciously, depending on the overall time taken by each step of the workflow and also on the extent to which each phase can be parallelized.

4. RELATED WORK

The performance of data analytics frameworks is an important concern. Pavlo *et al* compare map/reduce systems against distributed DBMSes and find interesting trade-offs in performance between these approaches [8]. They find that map/reduce is easier to setup but in the end the DBMS was more performant.

Ousterhout *et al* analyse real-life peta-scale workloads. They find that CPU is more often a bottleneck than I/O and that network performance has little impact on job completion time [7]. Moreover, they find that straggler nodes can be identified and that in most cases the cause for straggling can be identified.

Han *et al* perform a similar analysis for graph analytics frameworks [4]. They identified several opportunities for improvement in these systems. In a similar study, Satish *et al* [11] find that hand-optimized codes can outperform programmer-friendly frameworks by up to 560-fold.

Several authors have investigated analytics frameworks for shared-memory systems (single nodes), covering map/reduce workloads [9, 1] and graph analytics [12]. Kyrola *et al* optimize graph analytics assuming that the graph fits on disk but not in main memory [5] Zhang *et al* optimize graph analytics for non-uniform memory architectures [15].

5. CONCLUSION

As data analytics are applied to increasingly larger data sets, it is increasingly important to study and optimize the execution time of analytics operators. In this paper, we have studied in particular the impact of parallelism on the performance of data analytics, and in particular intra-node parallelism, which presents an important opportunity as Moore's Law remains valid.

Through studying one text processing and one numeric operator, we identified four optimizations related to intra-node parallelism that we expect are widely applicable across data analytics: intra-node parallel computation, parallel I/O, workflow optimization and selection of internal or intermediate data structures. We demonstrate that analytics queries have strong potential for performance optimization through

intra-node parallelism. Moreover, several optimizations, such as avoiding I/O through workflow fusion and choice of data structure, are influenced by the presence and degree of intra-node parallelism. This paper thus points out a new direction for realizing high-performance analytics and identifies open challenges.

6. REFERENCES

- [1] R. Chen and H. Chen. Tiled-mapreduce: Efficient and flexible mapreduce processing on multicore with tiling. *ACM Trans. Archit. Code Optim.*, 10(1):3:1–3:30, Apr. 2013.
- [2] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Çetintemel, and S. B. Zdonik. Tupleware: Big data, big analytics, small clusters. In *Conf. on Innovative Data Systems Research (CIDR)*, page 7, Jan. 2015.
- [3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.
- [4] M. Han, K. Daudjee, K. Ammar, M. T. Özsu, X. Wang, and T. Jin. An experimental comparison of pregel-like graph processing systems. *Proc. VLDB Endow.*, 7(12):1047–1058, Aug. 2014.
- [5] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *OSDI* pages 31–46, 2012.
- [6] Apache mahout: Scalable machine learning and data mining. <http://mahout.apache.org>.
- [7] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In *NSDI'15*, pages 293–307, 2015.
- [8] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD Intl. Conf. on Management of Data*, pages 165–178, 2009.
- [9] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *HPCA*, pages 13–24, 2007.
- [10] G. Salton and M. J. McGill, editors. *Introduction to Modern Information Retrieval*. Mcgraw-Hill, 1983.
- [11] N. Satish *et al*. Navigating the maze of graph analytics frameworks using massive graph datasets. In *SIGMOD Intl. Conf. on Management of Data*, pages 979–990, 2014.
- [12] J. Shun and G. E. Blelloch. Ligma: A lightweight graph processing framework for shared memory. In *ACM PPOPP*, pages 135–146, 2013.
- [13] Apache spark mllib. <http://spark.apache.org/mllib/>.
- [14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2–2, 2012.
- [15] K. Zhang, R. Chen, and H. Chen. NUMA-Aware graph-structured analytics. In *PPoPP* pages 183–193, 2015.