Sparse-matrix Arithmetic Operations in Computer Clusters: A Text Feature Selection Application

Antonela Tommasel[1], Cristian Mateos[2], Daniela Godoy[3] and Alejandro Zunino[4]

ISISTAN Research Institute, CONICET-UNCPBA
Tandil, Buenos Aires, Argentina
[1]antonela.tommasel@isistan.unicen.edu.ar
[2]cristian.mateos@isistan.unicen.edu.ar
[3]daniela.godoy@isistan.unicen.edu.ar
[4]alejandro.zunino@isistan.unicen.edu.ar

***Abstract*—Arithmetic operations on matrices are frequently used in scientific computing areas. They usually become a performance bottleneck due to their high complexity. In this context, the parallel processing of matrix operations in distributed environments arises as an important field of study. This work presents several strategies for distributing sparse matrix arithmetic operations on computer clusters, focusing on the intrinsic characteristics of the operations and the matrices involved. The performance of the proposed strategies for determining the number of parallel tasks to be executed on the computer cluster was evaluated considering a high-dimensional feature selection approach. Additionally, the performance of two alternatives for efficiently representing big-scale sparse matrices was tested. Experimental results showed that the proposed strategies significantly reduce the computing time of matrix operations, outperforming computations based on serial and multi-thread implementations.**

***Resumen*—Las operaciones aritméticas matriciales son frecuentemente utilizadas en muchas áreas de la computación científica. Usualmente, son un cuello de botella debido a su alta complejidad computacional. En este contexto, el procesamiento paralelo de operaciones matriciales en ambientes distribuidos surge como un importante tópico de estudio. Este trabajo presenta diversas estrategias para la distribución de operaciones aritméticas entre matrices ralas en *clusters* de computadoras. Dichas estrategias se basan en las características intrínsecas de las operaciones y las matrices involucradas. El desempeño de las estrategias propuestas para determinar la cantidad de tareas paralelas a ejecutar, fue evaluado considerando un enfoque de selección de atributos multidimensional. Además, se evaluó el desempeño de dos alternativas para la representación eficiente de matrices ralas a gran escala. Los resultados experimentales mostraron que las estrategias propuestas redujeron los tiempos de cómputo de las operaciones, mejorando así significativamente los tiempos de las implementaciones seriales y *multi-thread*.**

I. Introduction

The massive use of social media generates an extensive amount of data growing at an unprecedented rate. For example, *Facebook* has grown from 2,448 millions of posts published every day in 2012 to 4,750 millions of updates every day in 2013, which represents a 94% increase; *Twitter* has grown from 340 millions of tweets per day in 2012 to 500 millions of tweets in 2013, which represents a 47% increase. For text mining tasks, the feature space consisting of unique words or phrases that occur in those texts can comprise hundreds or thousands of features [1]. Thus, text learning is often susceptible to the problem known as curse of dimensionality, referring to the increasing computational complexity of problems as the data that need to be accessed grow exponentially regarding the underlying space dimension. Furthermore, as the data dimensionality increases, the volume of the feature space increases rapidly, so that the available data become sparse. The problem is aggravated if the linked nature of social media data is considered, causing new dimensions to be added to the feature space [2].

Large-scale text analysis, as the one needed in social environments, poses the challenge of efficiently processing high-dimensional data represented in a document-term matrix. Among the problems requiring the manipulation of such matrix is feature selection [3], a commonly used technique for dimensionality reduction. Feature selection aims at choosing a small subset of relevant features, excluding redundant and noisy features, from the original set according to certain evaluation criterion. Feature space reduction not only enhances predictive models but also helps to speed up data mining algorithms and to improve mining performance.

Simultaneously to the extensive amount of social data generated, scientific computing is constantly advancing. Single-processor architectures have evolved into multi-core architectures and, in recent years, Graphics Processing Units (GPUs) have emerged as co-processors capable of handling large amount of calculations. In this context, the parallel processing of matrix operations in distributed memory architectures arises as an important field of study [4]–[6]. In particular, the operations with dense matrices have been the subject of intensive research. However, the problem of operating with sparse matrices remains open.

This paper aims at studying the performance of several strategies for distributing sparse-matrix arithmetic operations on computer clusters. These strategies focus on the intrinsic characteristics of the operations and their associated matrices to compute a value called "parallel factor" that determines how matrix operations are partitioned to be processed on a computer cluster. The performance of the proposed strategies was evaluated considering the high-dimensional feature selection approach presented in [2]. Additionally, the extend to which the sparseness of matrices affects the performance of the proposed strategies is discussed.

The rest of this paper is organised as follows. Section II discusses related research. Section III presents different strategies to compute the parallel factor. Section IV describes the settings for the experimental evaluation carried

978−1−4799−4269−5/14/$31.00

out using the feature selection approach presented in [2] as potential application. Section V reports the results obtained. Finally, Section VI states the conclusions.

II. Related Works

Arithmetic operations on matrices are frequent in scientific computing areas, for example in signal and image processing [7], document retrieval [8] and feature selection [2], [9]. Those operations usually become a performance bottleneck due to their high computational complexity. In this context, the parallel processing of matrix operations in distributed memory architectures arises as an important issue to study. Particularly, operating with dense matrices has been the subject of intensive research [6].

In text analysis, as in collaborative filtering and document clustering, among others, matrices are sparse. Therefore, only the non-zero elements are stored, highlighting the importance of developing memory-efficient representations and algorithms. Notice that the performance of sparse matrix operations tends to be lower than the dense matrix equivalent due to the overhead of accessing the index information in the matrix structure and the irregularity of the memory accesses [5]. Additionally, algorithms that are efficient for dense representations are not suitable for sparse representations as often expend a large fraction of their computational resources performing unnecessary operations that involve zeros [10].

Several approaches have been developed for representing sparse matrices aiming at improving the efficiency of memory usage and computing operations. In [4] the focus is on reducing the number of accesses for particular matrix operations and defining a more natural mapping between the indices in the physical value matrix and the logical sparse coefficient matrix. In [5] the authors propose an approach to customise the matrix representation according to the specific sparseness characteristics of matrices and the target machine by performing register and cache level optimisations, for example by adding explicit zeros to improve the memory system behaviour. Results showed that the right choice of optimisations is essential to achieve performance improvements as each optimisation technique could benefit only a subset of matrices and negatively affect others.

In recent years, one of the strategies used to increase the computing power of computers has been the usage of Graphics Processing Units (GPUs) in addition to CPUs. Several works [11]–[13] have presented approaches for optimising matrix multiplications through the usage of GPUs. Both [11], [12] focused on the overlapping between computation and data communication in order to minimise the communication and transfer times. Dang et. al [11] presented an approach to efficiently represent sparse matrices and a CUDA (Compute Unified Device Architecture) implementation of matrix multiplications. The matrix representation was based on partitioning the input matrix according to its sparseness patterns. The matrix multiplication implementation considered a bi-directional data transfer. Oyarzun et al. [12] proposed an approach for matrix multiplication in the context of the conjugate gradient method in multi-GPU environments. In particular, the proposed solver was tested for the Poisson equation and reported improvements of up to a 200% regarding the CPU-only solver. Bell et al. [13] compared the performance of several matrix representations on their implementation of the matrix multiplication. Their implementation was tailored to the data access pattern of a particular GPU architecture to efficiently use the memory bandwidth and their results remarked the importance of choosing an adequate matrix representation according to its sparseness pattern.

Although most of the experimental evaluations of GPU-based approaches were performed on matrices with at most 1,000,000 rows and columns, the sparseness levels were higher than a 97%, thereby hindering the generalisation of results for smaller matrices with lower sparseness levels. Notice that smaller matrices with lower sparseness levels could require more resources than bigger matrices with higher sparseness levels. In conclusion, the experimental evaluations presented are not sufficient to correctly test the scalability of the proposed approaches, and do not accurately address the needs of document retrieval or feature selection problems.

In summary, GPU devices offer a considerable potential for performance and efficiency in large-scale applications of scientific computing. However, obtaining the desired performance from GPU devices is not a trivial task as usually requires significant changes in the algorithms and their implementations [12]. Furthermore, GPUs have limitations regarding the fixed memory capacity, maximum threads, memory access, and performance of any single device [13]. For example, the non-contiguous access to the GPU memory has a negative effect in the memory bandwidth efficiency, and thus, in the performance of memory-bound applications, specially when considering big data structures such as big-scale matrices [11]. Additionally, the distribution of tasks among GPUs presents some challenges as GPUs are unable to share memory space with CPUs or with each other [12]. In this context, this work proposes not only several strategies for efficiently distributing the parallel processing of arithmetic operations between matrices on computer clusters, but also tests two alternatives for efficiently representing big-scale sparse matrices.

III. Parallel Factor Computation Strategies

Considering the problems described before, this paper presents several strategies for distributing the parallel processing of arithmetic operations on computer clusters. These strategies are based on computing a Parallel-Factor (PF) to determine the appropriate number of rows of each task to be created, and thus the number of parallel tasks. The PF value is computed for three types of operations: addition/subtraction, matrix multiplication and Laplacian. Each type of operation has different information sharing requirements. Furthermore, the relevance of the involved matrices also varies. For example, in the case of matrix multiplications, it is important to consider the characteristics of the left matrix as its sparseness defines the number of actual multiplications that have to be performed, whereas in the case of adding or subtracting, both the sparseness of the left and right matrix are important. On the contrary, the Laplacian operation is independent of the sparseness of the matrix involved. Once it is computed, the PF is used to

determine the number of rows assigned to each parallel task to be created and executed on the computer cluster. The PF value is inversely related to the number of rows per tasks i.e., the bigger the PF, the lower the number of rows per tasks and thus, the bigger the number of tasks.

All the presented strategies but the first rely on the PF computation as defined in Eq. (1), and vary in the specification of α. Oppositely, the specification of γ as the ratio of the number of rows and columns, is shared by all the strategies as defined in Eq. (2). As the PF might be zero, which would assign zero rows to each task, an additional constraint is introduced. To prevent assigning zero rows to each task when the PF is zero, it is replaced by 1 to create at least one task.

$$\text{Parallel-Factor} = \lfloor \#\text{physical-cores} \times (1 + \alpha + \gamma) \rfloor \quad (1)$$

$$\gamma = \begin{cases} 1 - log\left(\frac{\#rows}{\#columns}\right) & \#rows \leq \#columns \\ 1 - log\left(\frac{\#columns}{\#rows}\right) & \#rows > \#columns \end{cases} \quad (2)$$

Static: This is the simplest strategy as it only depends on a static granularity factor as shown in Eq. 3. As the resulting PF is independent from the characteristics of the matrices involved, this is the strategy followed in all cases for the *Laplacian* operation. In this strategy, the bigger the granularity factor, the bigger the PF. The Granularity-Factor represents an alternative to manually force the creation of an arbitrary number of tasks.

$$\text{Parallel-Factor} = \lfloor \#\text{physical-cores} \times \text{granularity-factor} \rfloor \quad (3)$$

Row-Sparseness: As the general sparseness of a matrix could not accurately capture the sparseness of each particular row, this strategy considers the mean row sparseness of the matrix. The rationale behind this strategy is to establish an inverse relation between the PF and the row sparseness. As a lower row sparseness value would indicate high-populated rows, which can imply a higher number of operations to be performed, the number of rows per tasks should decrease. Eq. (4) shows how to compute the α value.

$$\alpha = 1 - \overline{\left(\frac{\text{non-zero}_i}{\#columns}\right)} \quad (4)$$

Row-Sparseness Standard-Deviation (Row-Sparseness-SD): Since all data points are used to compute the mean of a distribution, outliers can affect the accuracy of results. This strategy aims at considering the standard deviation of the data distribution in order to add information regarding the existence of outliers. The standard deviation (σ) measures the dispersion of data from the mean. A low σ indicates that the data points are close to the mean, whereas a high standard deviation indicates that the data points are spread over a large range of values. As Eq. (5) shows, α considers the difference between the mean and the standard deviation, i.e. the lowest sparseness value in the normal distribution.

$$\alpha = 1 - \left(\overline{\left(\frac{\text{non-zero}_i}{\#columns}\right)} - \sigma\left(\frac{\text{non-zero}_i}{\#columns}\right)\right) \quad (5)$$

Mode: The mode of a data distribution can be defined as the most frequent value, which in this case can be defined as the most frequent row sparseness value, as shown in Eq. (6). Notice that the mode of a distribution may not accurately represent the data as there may be more than one mode value, or no value at all if no value occurs more than once. Additionally, in case the mode exists, it favours the most common sparseness value, which could cause an unbalanced distribution of rows per task in terms of the resulting sparseness, affecting the real parallelism of the task.

$$\alpha = 1 - \text{most-frequent}\left\{\frac{\text{non-zero}_i}{\#columns}\right\} \quad (6)$$

IV. Experimental Settings

The proposed strategies performance for computing PF, and thus, determining the number of parallel tasks to be created, was evaluated for the feature selection approach of [2]. Interestingly, the feature selection approach considers not only features and posts, but also the social context of the posts and the relationship between users, producing high-dimensional matrices representing texts and social contexts. According to the authors, social media data analysis and social correlation theories such as homophily and social influence, suggests the existence of four types of relations: *Co-Post* (posts written by the same user share the same topic), *Co-Following* (if two users follow the same user, their posts are likely to be topically related), *Co-Followed* (if two users are followed by the same user, their posts are likely to be topically related) and *Following* (relations between users are based on the relatedness of the post's topics). For each type of relation, the authors proposed a hypothesis to study how they affect the feature selection process, which were based on big-scale matrices and arithmetic operations between them. The feature selection process included the computation of specific matrices according to the chosen hypothesis, and then, the iterative modification of data until convergence is reached. Complete definitions, mathematical proofs and algorithm can be found in [2].

This study focuses on the first part of the feature selection algorithm, i.e. the definition of the matrices specific to each hypothesis. In particular, this study focuses on the *Co-Post* relation, which involves the definition of two matrices, denoted B and E:

$$B = XX^T + \beta F L_A F^T \quad (7)$$

$$E = Y^T X^T = (XY)^T \quad (8)$$

, where β represents the impact of the social relation on the feature selection process, F represents the set of features for a post where $F_{i,j}$ is the frequency of $feature_j$ in $post_i$, X represents a subset of F in which only the labelled post are considered, Y represents the class label matrix for labelled data where $Y_{i,j} = 1$ if $post_i$ is labelled as $class_j$ and zero otherwise, and L_A represents the Laplacian matrix of $A = P^T P$ where P represents the set of posts of a user and then $A_{i,j} = 1$ is $post_i$ and $post_j$ were posted by the same user and zero otherwise. Table I shows the list of individual operations involved in computing B and E, which were the focus of the experimental evaluation. Note that

Table I
LIST OF INDIVIDUAL OPERATIONS INVOLVED IN COMPUTING B AND E

B	Matrix Multiplication I	$A = P^T P$
	Addition-Subtraction I	$D_{A_{i,i}} = \sum_j A_{j,i}$
	Addition-Subtraction II	$L_A = D_A - A$
	Matrix Multiplication II	$\beta F L_A$
	Matrix Multiplication III	$\beta F L_A F^T$
	Matrix Multiplication IV	XX^T
	Addition-Subtraction III	$XX^T + \beta F L_A F^T$
E	Matrix Multiplication V	$(XY)^T$

the Laplacian computation is separated into three individual operations: *Matrix Multiplication I*, *Addition-Subtraction I* and *Addition-Subtraction II*.

Java was the programming language chosen for implementing the approach [14] as it was shown that it can achieve a similar performance to optimised languages such as Fortran [15]. Matrices were implemented as sparse memory structures in order to decrease the storage and network transfer requirements. The performance of two internal representations of matrices was evaluated. First, matrices were implemented using the *HashMap* structure provided by Java, which only stores objects (non-primitive type elements) incurring in boxing and un-boxing operations each time an element is retrieved from the structure. On the other hand, matrices were implemented using the *Trove*[1] implementation of Hash structures, which stores primitive types elements avoiding boxing and un-boxing. Although the double type provides more precision in operations than the float type, the latter was chosen as the type of the matrices' elements due to the space required to store a matrix full of double elements.

The distribution and execution of tasks on the computer cluster was performed by using the Java Parallel Processing Framework (JPPF)[2] middleware, which allows applications to be executed on any number of computers. JPPF follows a master-slave design in which a server distributes the tasks of a job among an arbitrary number of nodes for parallel execution. In order to reduce network transfer times and data duplication, matrices shared by more than a task belonging to the same job were not duplicated in the node. Further reductions were achieved by using the local class loading option in nodes.

The baseline for comparing and evaluating the enhancements introduced by using the proposed parallel factor to distribute tasks on the cluster was the execution of all the operations in a serial and a multi-thread manner. In particular, for the multi-thread execution pools of 3, 4 and 5 threads were considered (namely *Multi-Thread-3*, *Multi-Thread-4* and *Multi-Thread-5*). All the serial and multi-thread executions were performed on a i7-3820 processor running at 3.6 GHz with 32 GB RAM. On the other hand, the computer cluster comprised 5 computers having an AMD Phenom II X6 1055T processor running at 2.8 GHz with 8 GB RAM, and 3 computers having an AMD FX-6100 running at 3.6 GHz with 16 GB RAM. All computers were connected by means of a Gigabyte network. All the presented strategies were executed at least five times, so that the mean values are reported. The standard deviation was lower than a 0.06% in all cases.

[1]http://trove.starlight-systems.com/

[2]http://www.jppf.org/

Table II
DIGG COLLECTION MAIN CHARACTERISTICS

Number of Posts	42,843
Number of Features	8,546
Number of Classes	51
Number of Following Relations	56,440
Minimum number of Followees	4
Maximum number of Followees	622
Minimum number of Posts per User	1
Maximum number of Posts per User	1,101
Minimum number of Features per Posts	1
Maximum number of Features per Posts	10
Minimum number of Post per Class	38
Maximum number of Post per Class	4,280
Average number of Following Relations	157
Average number of Features per Post	4
Average number of Posts per Class	840

Table III
MATRICES' SIZE AND SPARSENESS PER OPERATION

	Operation	*Size*	*Sparseness*
B	Matrix Multiplication I	42,843x42,843	99.55%
	Addition-Subtraction II	42,843x42,843	99.54%
	Matrix Multiplication II	8,546x42,843	94.80%
	F^T	42843x8546	99.96%
	Matrix Multiplication III	8,546x8,546	66.94%
	X^T	42843x8546	99.96%
	Matrix Multiplication IV	8,546x8,546	99.47%
	Addition-Subtraction III	8,546x8,546	66.95%
E	Matrix Multiplication V	8,546x51	66.94%

The data collection created in [16] was used in the experimental evaluation of the presented approach. The collection comprised data extracted from *Digg*[3] in August 2008. *Digg* is a social news website that allows its users to share and comment content (posts), and to vote the content up or down. In addition, users are encouraged to establish social relationships by adding other users to their friend network and following their activity. The data collection includes information about posts, authorship of posts, the topic of each post, which is considered as the class of the post, comment and voting activity on each post, and non-reciprocal social relationships between users. In this dataset, posts were already pre-processed to remove stopwords. For the purpose of the experimental evaluation, the comment and voting activity on each post was discarded. Table II summarises the main characteristics of the dataset.

V. EXPERIMENTAL RESULTS

This section reports the results for the four baseline executions and the four novel strategies presented. In the case of the *Static* strategy, two values for the Granularity-Factor were adopted, 1 (named *Static-1*) and an arbitrary value of 30 (named *Static-30*), totalling nine different evaluation cases. Table III shows the sparseness level of the result matrices of each of the evaluated operations and several auxiliary matrices. As it can be observed, the sparseness of the matrices ranged between 67% and 99%. The most dense matrix corresponded to the result of a matrix multiplication, which is the most resource demanding operation.

[3]http://www.digg.com/

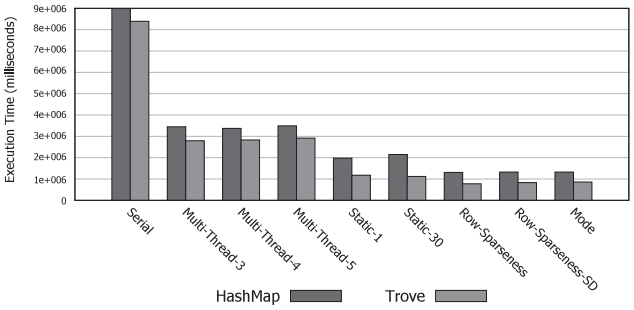

Figure 1. B Matrix Overall Computing Time (logarithm scale)

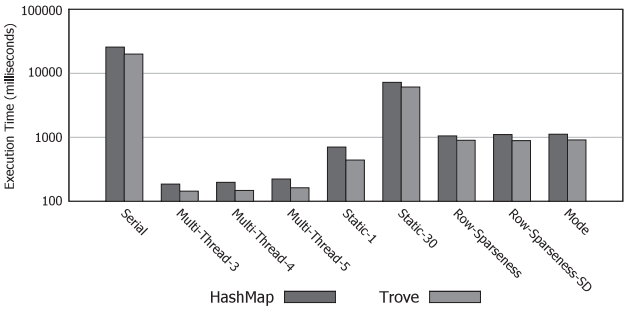

Figure 2. E Matrix Overall Computing Time (logarithm scale)

Figure 1 and Figure 2 compare the results obtained for the matrix representations tested for both B and E matrices, as defined in Eq (7) and Eq. (8). The results include the data transference and storage times, and the computing time of the PF, when applicable. As it can be observed, the performance of the *Trove* based representation was superior to the *HashMap* based representation in all cases, improving results from a 7% when considering the *Serial* execution, up to a 48% when considering the *Static-30* strategy regarding the computation of B. Additionally, it can be observed the improvements in execution times caused by the usage of the proposed strategies, with differences up to a 91% when considering the best strategy (*Row-Sparseness*) regarding the *Serial* implementation. Considering that the experiments only varied in the matrix representation used, the performance difference could be explained in terms of different network transfer and storage times, and thus, it can be stated that the different matrix representations have different storage needs. In this regard, Figure (3) shows the size of the matrices involved in the operations presented in Table III. The *Trove* based representation required less storage space than the *HashMap* based representation, with differences ranging from a 11% for the case of the F^T needed for computing *Matrix Multiplication III,* up to a 59% in the case of the $A = P^T P$ operation needed for computing the Laplacian. These results imply that *Trove* uses a more efficient internal structure.

Considering the previous results, Figure 4 presents the execution times of each operation for the nine evaluation cases using the *Trove* based representation. As the figure shows, the strategies based on executing on the computer cluster achieved the best results for most of the operations. The only exception was the *Addition-Subtraction III* operation. The matrix corresponding to that operation was

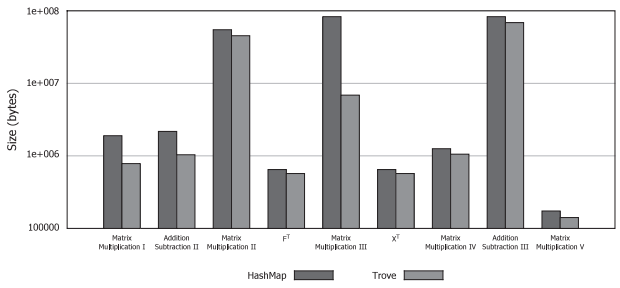

Figure 3. Matrix Size Comparison (logarithm scale)

one of the least sparse ones (66%), and consequently, the one that required more accesses to elements in order to perform the operation. Additionally, more data needed to be transferred as rows were also less sparse, which negatively affected the computer cluster executions. Conversely, when considering the *Addition-Subtraction II* operation, the best results were achieved by the computer cluster executions. An analysis of the result matrix showed that the sparseness level was close to 99%. These results confirmed the impact that the sparseness level has in the performance of operations as it affected network transfer and storage requirements. In summary, the computer cluster executions outperformed the best performing *Serial* and *Multi-thread* executions by a 56% in the most time consuming operation (*Matrix Multiplication III*) and 98% in the best case (*Addition-Subtraction II*) when considering the *Row-Sparseness* and *Mode* strategies for computing B respectively. However, when the computations involved small matrices such as the operations involved in E, *Multi-thread* executions tended to perform better than the computer cluster ones. This could be also explained in terms of network transfer times, which negatively affected the performance, as most part of the time was spent on transferring data instead of performing the actual computations.

Finally, as regards the computer cluster executions solely, the worst overall results were obtained for the *Static-1* strategy, stating the importance of considering the intrinsic characteristics of the operations to be performed, and the characteristics of the matrices involved. Oppositely, the best overall results were obtained for the *Row-Sparseness* strategy, outperforming the worst strategy by a 51%, and the second best by a 7%. As regards the individual operations, *Row-Sparseness* also achieved the best results for most of them with differences ranging between 5% and 55% in the most time consuming operation (*Matrix Multiplication III*), reinforcing the importance of considering the matrix characteristics when computing PF, particularly the mean row sparseness. Additionally, these results also highlight the fact that the time spent computing the α value for the *Row-Sparseness*, *Row-Sparseness-SD* and *Mode* strategies was statistically insignificant, representing less than a 0.03% of the total time of each operation, and thus, it did not affect the overall performance of the approach.

VI. Conclusions

This work aimed at studying the performance of several strategies for distributing sparse matrix arithmetic operations on computer clusters. The strategies focused on the

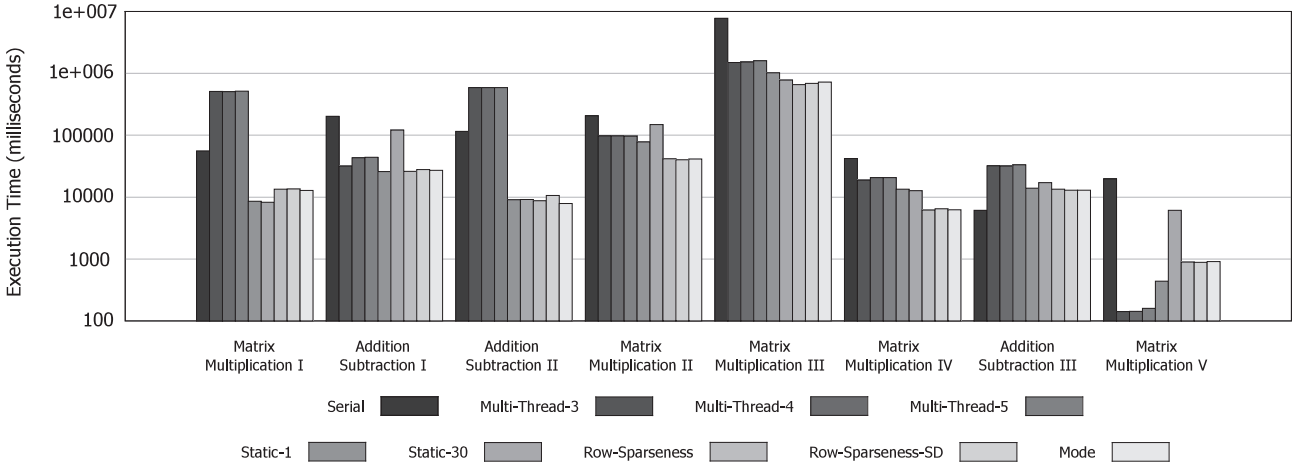

Figure 4. Computing Time of all Individual Operations (logarithm scale)

intrinsic characteristics of the operations and their associated matrices. The performance of the proposed strategies for computing the PF to determine the number of parallel tasks to be created was evaluated considering the high-dimensional feature selection approach presented in [2].

Experimental evaluation showed that the performance of the *Trove* representation of matrices was superior to the *HashMap* one as *Trove* uses a more efficient internal structure requiring less network and storage resources. Regarding the overall computing times, the computer cluster executions outperformed the *Serial* and *Multi-Thread* executions when big-scale matrices were involved. On the contrary, when the operations involved small matrices, the *Multi-Thread* executions tended to perform better than the computer cluster ones. This could be explained in terms of the network transfer and communication times as most part of the time was spent on transferring data instead of performing the actual computations.

Regarding the strategies proposed, the worst overall results were obtained for the *Static* strategy as expected. The best overall results were obtained for the *Row-Sparseness* strategy, which also outperformed the other strategies for most of the individual operations. These results stated the importance of considering the intrinsic characteristics of the operations to be performed and the characteristics of the matrices involved, in particular the mean row sparseness. Finally, the results also highlighted the fact that the time spent computing the PF was statistically insignificant, not affecting the overall performance of the strategies.

REFERENCES

[1] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, (San Francisco, CA, USA), pp. 412–420, Morgan Kaufmann Publishers Inc., 1997.

[2] J. Tang and H. Liu, "Feature selection with linked data in social media.," in *SDM*, pp. 118–128, SIAM / Omnipress, 2012.

[3] S. Alelyani, J. Tang, and H. Liu, "Feature selection for clustering: A review.," in *Data Clustering: Algorithms and Applications*, pp. 29–60, 2013.

[4] K. Zhang and B. Wu, "Parallel sparse matrix multiplication for preconditioning and ssta on a many-core architecture," in *Proc. Int. Conf. on NAS'12*, pp. 59–68, June 2012.

[5] E.-J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization framework for sparse matrix kernels," *Int. J. High Perform. Comput. Appl.*, vol. 18, pp. 135–158, Feb. 2004.

[6] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Trans. Math. Softw.*, vol. 39, pp. 13:1–13:24, Feb. 2013.

[7] H. Liu, J. He, D. Rajan, and J. Camp, "Outlier detection for training-based adaptive protocols," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pp. 333–338, April 2013.

[8] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, (New York, NY, USA), pp. 267–273, ACM, 2003.

[9] Z. Zhao, L. Wang, and H. Liu, "Efficient spectral feature selection with minimum redundancy.," in *AAAI* (M. Fox and D. Poole, eds.), AAAI Press, 2010.

[10] L. Heath, C. Ribbens, and S. Pemmaraju, "Processor-efficient sparse matrix-vector multiplication," *Computers & Mathematics with Applications*, vol. 48, no. 34, pp. 589 – 608, 2004.

[11] H.-V. Dang and B. Schmidt, "Cuda-enabled sparse matrix-vector multiplication on gpus using atomic operations," *Parallel Computing*, vol. 39, no. 11, pp. 737 – 750, 2013.

[12] G. Oyarzun, R. Borrell, A. Gorobets, and A. Oliva, "Mpi-cuda sparse matrix-vector multiplication for the conjugate gradient method with an approximate inverse preconditioner," *Computers & Fluids*, vol. 92, no. 0, pp. 244 – 252, 2014.

[13] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on cuda," NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008.

[14] G. L. Taboada, S. Ramos, R. R. Expósito, J. Touriño, and R. Doallo, "Java in the high performance computing arena: Research, practice and experience.," *Sci. Comput. Program.*, vol. 78, no. 5, pp. 425–444, 2013.

[15] J. E. Moreira, S. P. Midkiff, M. Gupta, P. V. Artigas, P. Wu, and G. Almasi, "The ninja project," *Communications of the ACM*, vol. 44, pp. 102–109, Oct. 2001.

[16] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher, "Metafac: community discovery via relational hypergraph factorization," in *Proc. ACM SIGKDD*, KDD '09, pp. 527–536, 2009.