Developing a Protein Interaction Prediction Algorithm on HPC

Wael S. Afifi[#1], Ali A. El-Moursy[*2], Prof. Salwa Nassar[#3]
[#]Computer and Systems, Electronics Research Institute
Cairo, Egypt
[1]waelsafifi17@eri.sci.eg
[3]salwa@eri.sci.eg
[*]Electrical and Computer Eng., University of Sharjah
Sharjah, UAE
[2]aelmorsi@sharjah.ac.ae

Prof. Mohy Abo-elsoud[*4], M. A. Mohamed[*5]
[*]Electronics and Communication Eng., Mansoura University
Mansoura, Egypt
[4]mohyldin@yahoo.com
[5]mazim12@yahoo.com

***Abstract*— The prediction of protein-protein interaction is one of the fundamental problems in bioinformatics. A novel algorithm called STRIKE has shown to achieve good performance in protein-protein interaction prediction. It assumes that proteins interact if they contain similar substrings of amino acids. In this paper, we developed a parallel STRIKE algorithm and we implemented our proposal on Cluster system. Using short protein sequence sets, the overall execution time of a parallel implementation of this bioinformatics algorithm was decreased to about 5 times when increasing number of nodes from one compute node to 6 parallel nodes. Key optimizations to the implementation are also discussed.**

***Keywords*— protein-protein sequence matching; parallel computing; performance analysis; HPC computing; sequence comparison**

I. INTRODUCTION

The prediction of protein-protein interaction (PPI) is one of the fundamental problems in computational biology as it can aid significantly in identifying the function of newly discovered proteins. Understanding protein-protein interactions is crucial for the investigation of intracellular signaling pathways, modeling of protein complex structures and for gaining insights into various biochemical processes.

To solve this problem, many experimental techniques have been developed to predict the physical interactions which could lead to the identification of the functional relationships between proteins. These experimental techniques are however, very expensive, significantly time consuming and technically limited, resulting in a growing need for the development of computational tools that are capable of identifying PPIs. To this end, many impressive computational techniques have been developed. Each of these techniques has its own strengths and weaknesses, especially with regard to the sensitivity and specificity of the method. Some of the state-of-the-art techniques such as the Association Method (AM) [1], Maximum Likelihood Estimation (MLE) [2], Maximum Specificity Set Cover (MSSC) [3] and Domain-based Random Forest [4] have employed domain knowledge to predict PPI. The motivation behind this employment is that molecular interactions are typically mediated by a great variety of interacting domains. PIPE (Protein-Protein Interaction Prediction Engine) [5] was also developed and it is based on the assumption that some of the interactions between proteins are mediated by a finite number of short polypeptide sequences. These sequences are typically shorter than the classical domains, and are used repeatedly in different proteins and contexts within the cell.

However, identifying domains or short polypeptide sequences is a long and computationally expensive process.

These techniques are also not universal because the accuracy and reliability of these methods is dependent on the domain information of the protein partners.

In this paper, we introduce a novel algorithm termed STRIKE which employs String Kernel to predict PPI. The string kernels (SK) approach has been shown to achieve good performance on text categorization tasks [6] and protein sequence classification [7]. The basic idea of this approach is to compare two protein sequences by looking at common subsequences of fixed length. The string kernel is built on the kernel method introduced by [8] and [9]. The kernel computes similarity scores between protein sequences without ever explicitly extracting the features. A subsequence is any ordered sequence of amino acids occurring in the protein sequence, where the amino acids are not necessarily contiguous. The subsequences are weighted by an exponentially decaying factor of their full length in the sequence, hence emphasizing those occurrences that are more contiguous. We understand that the subsequences' similarity between two proteins may not necessarily indicate interaction, however, it is evidence that we can't ignore. Subsequence similarity helps in inferring homology. Homologous sequences usually have the same or very similar structural relationships.

A drawback of this approach is observed when the level of similarity between the protein pairs is too low to pick up interaction. The reasonable explanation is that in the case of low sequence, there are always similar patterns of identical amino acid residues which could be seen in the two sequences. The pattern of sequence similarity reflects the similarity between experimentally determined structures of the respective proteins or at least corresponds to the known key elements of one such structure [10]. Structural evidence indicates that, interacting pairs of close homologs usually interact in the same way [11]. The Likelihood ratio in this study expresses the reliability of such genomic feature. In our case, there is no doubt that the SK method is a good indicator of homology between protein pairs. The intensive comparison between subsequences exists in protein pair may capture structural domain knowledge or typically subsequences which are shorter than the classical domains and could appear repeatedly in the protein pairs of interest. We are also encouraged by the success of a recently published work employing pairwise alignment as a way to extract meaningful futures to predict PPI. The PPI based on Pairwise Similarity (PPI-PS) method consists of a representation of each protein sequence by a vector of pairwise similarities against large subsequences of amino acids created by a shifting window which passes over concatenated protein training sequences. Each coordinate of this vector is typically the E-value of the Smith-Waterman score [12]. One major drawback of the PPI-PS is that each protein is represented by computing the Smith-Waterman score against a large subsequence created by concatenating protein training sequences. However, comparing short sequences to very long ones will result in some potentially valuable alignments to be missed out. The SK however, tackles this weakness by capturing any match or mismatch which exists in the protein sequence of interest.

In Section 2, we explain the parallel protein sequence decomposition and matching algorithm on parallel nodes. Section 3 formally presents the parallel sequence matching algorithm and its complexity analysis, and discusses the implementation of the application on multiple nodes. Section 4 reports HPC platforms used in our experiments. Section 5 presents the resulting performance analysis and results and. The paper concludes in Section 6.

II. Parallel Protein Sequence Matching Algorithm

We start explaining how the algorithm works by a simple example which compares the two short protein sequences s1="lql" and s2="lqal", where there exists one string of characters in each sequence. For computational simplicity and to meet common memory capacities of modern computers, we set the length of substring (patterns to match) to 2. In other words, these sequences are implicitly transformed into feature vectors, where each feature vector is indexed by the substrings of length 2. Table I shows the decomposition of each of the two sequences into 2-character substrings. Each sequence is decomposed into all possible ordered (from left to right) combinations of characters included in the sequence such that the 2 characters need not be consecutive. The first three (from the left) 2-character substrings represent the decomposition of the s1 sequence, while all 6 2-character substrings represent the decomposition of the second sequence s2.

TABLE I
Mapping Two strings "LQL" and "LQAL" to Six Dimensional Feature Space

	lq	**ll**	**ql**	**la**	**qa**	**al**
S1 = Ø(lql)	λ^2	λ^3	λ^2	0	0	0
S2 = Ø(lqal)	λ^2	λ^4	λ^3	λ^3	λ^2	λ^2

When a 2-character substring appears in a sequence such that these 2 characters are consecutive in the sequence, the substring's *dom* –degree of matching— in that sequence is represented by λ^2, where λ is a decay factor. For instance, the substring "lq" fits this case in the first sequence. When these 2 characters are separated by another character (gap of 1), the substring's *dom* is $\lambda^{2+\text{gap of } 1} = \lambda^3$. The substring "ll" fits this case in the first sequence. When the 2 characters in the appearing substring are further spaced by exactly gap characters, the *dom* is represented by $\lambda^{2+\text{gap}}$. The *doms* for the substrings for the first sequence and all substrings for the second sequence are computed in that fashion as shown in Table I. When matching the 2 sequences, the 2-character substrings to impact and increase the degree of matching must exactly appear in both sequences.

To reflect the degree of matching between the s1 and s2 sequences, the un-normalized string kernel (SK) for the 2 sequences, k(lql,lqal) can clearly be computed as the dot product of the 2 rows of Table 1 containing the *doms*, i.e. $\lambda^4+\lambda^7+\lambda^5$. Assuming that the decay factor λ is equal to 0.5, k(lql,lqal)=0.102. The higher the un-normalized kernel the higher the indication of matching between the 2 sequences and the higher is the interaction.

To parallelize this algorithm, we describe a highly parallel algorithm consisting of the following 3 steps:

i. Decomposition
ii. Sorting
iii. Inner Product

In the decomposition step, the amino acid sequences are allocated to processing nodes, one sequence per node. For instance let us assume that the SKs of the 4 amino acid sequences "lyq," "qyla", "yqla" and "qla" are computed on four parallel computing nodes. The goal is to find mutual interaction between these 4 sequences.

The processing node allocation of the 4 sequences proceeds as shown in Fig. 1.a.

In all nodes, the decomposition of each protein sequence proceeds in parallel and their execution times overlap in time. Each sequence is decomposed into 2-amino acid substrings starting with adjacent amino acids as shown in Fig. 1.b. The "2" in "(ly 2)" refers to the power of the weighted decay factor (λ) (i.e. λ^2) indicating no gap (i.e. 3rd character) between the "l" and the "y."

Since the amino acids in the resulting substring are not necessarily required to be contiguous, the decomposition into 2-amino acid substrings with a non-adjacent amino acid separated by another amino acid takes place as illustrated in Fig. 1.c.

Again, the "3" in "(lq 3)" refers to the power of the weighted decay factor (λ) (i.e. λ^3), meaning that the "l" and "q" are separated by another amino acid ("y") in the sequence "lyq". Finally the decomposition into 2-amino acid substrings composed of non-adjacent amino acid separated by 3 other amino acids takes place as shown in Fig. 1.d.

As nodes 1 and 4 have shorter sequences to process in step 1, they will complete step 1 ahead of processing nodes 2 and 3. Thus nodes 1 and 4 can immediately proceed to step 2, while nodes 2 and 3 will proceed to step 2 immediately after completing step 1.

In the second step of the parallel algorithm, the 2-amino acid substrings generated in the first step are sorted alphabetically based on their 2-letter string content. Again each node sorts its strings alphabetically in parallel with the other nodes so the string sortings in all 4 nodes overlap in time. After step two completes, the 4 processing nodes will have for content the sorted strings shown in Fig. 1.e.

In the third step, the inner products are carried out on half
(4/2=2) the nodes with the largest substring set cardinality. This choice is made to minimize the total inter-node communication time. In our example, nodes 2 and 3 have the highest number of generated substrings. Each of these nodes maintains its 2-amino acid substrings and receives 3 amino acid strings generated by the node which is allocated the other sequence to match with its sequence. To simplify this example, let us say that our goal is to match the protein sequences "lyq" (node 1) and "qyla" (node 2) together, and the protein sequences "yqla" (node 3) and "qla" (node 4) together, in step 3, and not all the 4 sequences with each other. As a result, the following data communications will take place, as shown in Fig. 1.f.

Node 1 sends its generated 2-amino acid substrings to node 2, and node 4 sends its generated 2-amino acid substrings to node 3.

In our case of a message-passing system, the data communication takes place in the form of messages sent by the sender nodes (1 and 4) to the destinations nodes (2 and 3).

In case of a shared memory system [13], processing nodes 2 and 3 read the 2-amino acid substring data generated by nodes 1 and 4 from shared memory. After the data is received or read by the destination nodes, the processing nodes 2 and 3 will hold the substrings shown in Fig. 1.f. nodes 1 and 4 need not remain active.

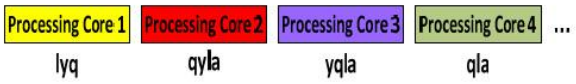

Fig. 1.a Allocation of sequences to processing cores (nodes)

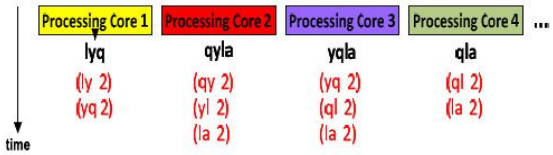

Fig. 1.b Decomposition of each protein sequences into substrings of Length=2 and Distance=1

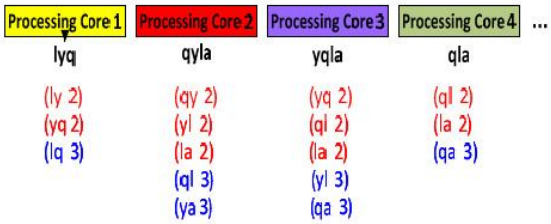

Fig. 1.c Decomposition into substrings of Distance=2

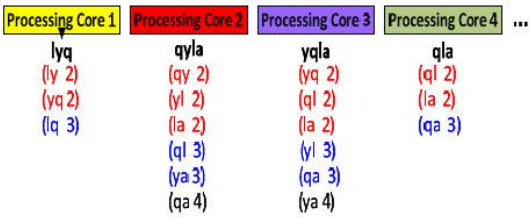

Fig. 1.d Decomposition into substrings of Distance=3

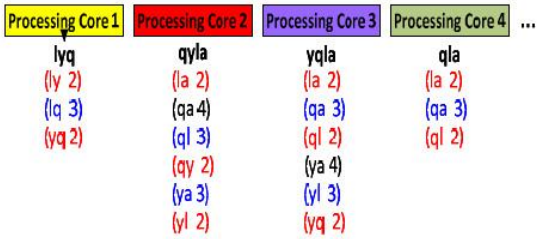

Fig. 1.e Content of the cores (nodes) is sorted substrings

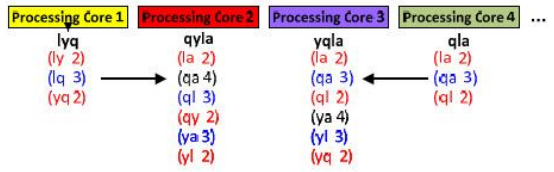

Fig. 1.f Inter-core (Inter-node) communication

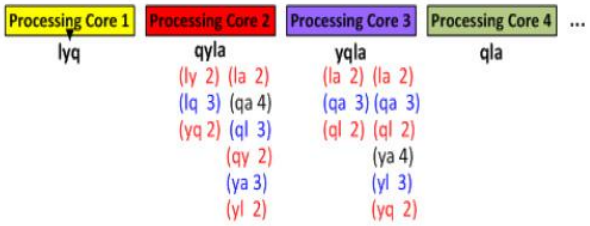

Fig. 1.g Contents of each core (node) after inter-core (inter-node) communications

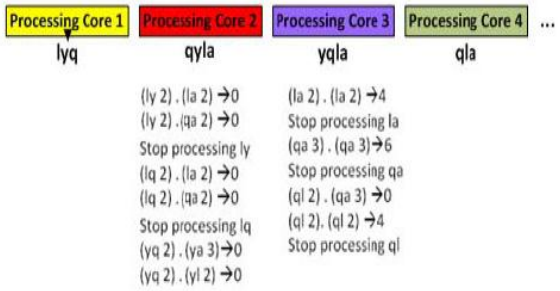

Fig. 1.h Inner products

In our example, node 2 and 3 then start performing the inner products between their strings generated in step 2 and the received strings generated by the neighboring node are shown in Fig. 1.g. The inner product (α n) . (β m) succeeds when the 2 strings match i.e. α=β, producing the number n+m (representing λ^{n+m}). Otherwise if α is different from β, then it's a mismatch (resulting in 0). Thus nodes 2 and 3 will simultaneously perform the following inner products. Note that node 2 will take the product of ly (followed by lq, and yq, respectively) with the substrings in the other set. The results are presented by each node involved in the inner product step as follows:

Node	**Result**
2	0
3	4,6,4: $\lambda^4 + \lambda^6 + \lambda^4 = 2\lambda^4 + \lambda^6$

On a processing node, matching two substrings starting with the same amino acid will speed up the kernel computation step. After matching a string with all other substrings starting with the same amino acid, the remaining strings in the second sequence can be skipped as the strings have been sorted in alphabetical order in the second step. For instance, referring to above results, after matching (ly 2) to (la 2), processing of the string (ly 2) stops as the remaining strings in the second set do not start with the amino acid l. This could be implemented by a simple indexing mechanism based on the starting amino acid of the substrings. In the absence of such mechanism, a string will have to be matched (i.e. its inner product taken) with all strings in the other set until a match is found or until all the strings in the other set have been exhausted.

Step 3 can be repeated as many times as needed to match other protein sequences allocated to other processing nodes. For instance to match the lyq (allocated to processing node 1) and qla (allocated to processing node 4) sequences, processing node 4 sends its 2-amino acid substrings generated in step 2 to processing node 1 which carries out the inner product step. Thus the parallel algorithm is capable of matching as many sequences in parallel as desired based on the availability of processing nodes. STRIKE is highly parallel and should achieve excellent performance scalability with increasing hardware resources.

III. MPI Parallel Implementation on HPC

In our message-passing interface (MPI) implementation, we make two changes to the previous algorithm. For efficiency and proper indexing, we skip the sorting step and perform matching between all the 2-character substrings of the two protein sequences to match. Second, to improve the matching accuracy, we modify the SK to be the weighted inner product of the doms (α n) . (β m), i.e. from λ^{n+m} to λ^{n+m} x matrix(c1) x matrix(c2), where c1 and c2 are the first and second characters appearing in the matching substrings α=β="c1 c2", and matrix(c1) is a weight given to characters, such that alphabetical characters A, B, …C can be assigned different weightage helping direct the matching towards character-orientation.

The parallel implementation consists of a main procedure set and the other from the testing set, and amino acid matrix, which reads the input protein sequences, one from the training launches parallel jobs which are assigned an equal number of sequences to match and which generate the pairs of amino acids and their inter-distances and compute the portion of the score matrix corresponding to the sequences assigned to these jobs. The matrix contains all amino acid weights corresponding to all characters in the protein sequence. Afterwards, control is passed back to the main procedure for printing the score matrix, and computing the execution time. The basic STRIKE procedure proceeds as follows.

Algorithm STRIKE

Data: - Files train.txt, test.txt containing the protein Sequences

- File matrix.txt containing the weights of each amino acid in the training sequence

- value of lambda, λ, 0.8 by default

- *NumThreads*, number of parallel threads to launch

Begin

1. Read all amino acid sequences from their data files. The training sequences were read from a train.txt file, while the testing sequences were read from a test.txt file, while the matrix values assigning each of the amino acid characters a weight were read from a matrix.txt file.

2. **For each** sequence in the training and testing sets, pair each amino acid with one subsequent amino acid, and store this pair of amino acids along with the distance between these 2 amino acids.

3. Launch *numnodes* **parallel** jobs with an equal load of sequences each performing the following sequential steps

3.1 Set the matching $score_{i,j}$ corresponding to the 2 amino acid sequences *i* and *j* to 0.

3.2 **For any** 2 amino acid sequences, match the pairs of amino acids of the first sequence *i* in the testing set with the pairs of amino acids of the second sequence j in the training set.

3.3 **If** both amino acid pairs exactly match, **then**

3.3.1 add their distances together,

$$dist_{i,j} = dist_i + dist_j$$

3.3.2 update $score_{i,j}$ as follows

$$score_{i,j} = score_{i,j} + \lambda dist_{i,j} \times matrix(amino\ acid_i) \times matrix(amino\ acid_j),$$

where matrix(*a*) is the matrix value corresponding to amino acid letter "*a*."

4. When all jobs are **done** all pairs of sequences assigned to them, communicate to gather score in one node at least, then print the sequence $score_{i,j}$'s as a matrix of floating point numbers with row index i and column index j. Also, calculate and print the execution time.

End

Complexity-wise, step 1 is $O(l_1 \times n_1 + l_2 \times n_2)$ where n_1 and n_2 are the lengths of the protein sequences, and l_1 and l_2 are the numbers of protein sequences in both testing and training sets. Step 2 is $O(l_1 \times n_1^2 + l_2 \times n_2^2)$. Step 3 is $O(l_1 \times l_2 \times n_1 \times n_2)$. Step 4 is $O(l_1 \times l_2)$. Therefore the entire algorithm is $O(l_1 \times l_2 \times n_1 \times n_2)$.

IV. Experimental Setup

The application was implemented on two different high performance computing (HPC) platforms or clusters; the main difference between them is the scaling that we could reach to in both. The small-scale cluster has 10 PCs, one acts as the server, while the others are the clients. The server has 2 single core processors. Each processor is 3.4 GHz Intel Pentium 4 CPU, while the clients are classified as follows:

i. Three clients of single core processor, each processor is 3.4 GHz Intel Pentium 4 CPU.

ii. Six clients of two dual core processors, each processor is 2.4 GHz Intel Pentium Dual CPU.

The cluster nodes are connected via a 3Com LAN switch 10/100/1000 MHz and LAN cables of types CAT5 (which enable data rates up to 100 MHz) and CAT5e (which enable data rates up to 1000 MHz).

We used gcc complier, version 3.4.6 20060404 (Red Hat 3.4.6-8) which has a lot of optimized implementations for the different libraries on Linux families.

For parallel experiments, we used MPICC compiler, version mpich-1.2.4 which is a freely available, portable implementation of MPI (Message Passing Interface) used to allow computers to communicate with each other.

Our experiment is done on various number of nodes (i.e., 1, 2, 4 and 6 nodes) using short-sequence set of data. Performance Analysis is done and explained in next section.

V. Results and Analysis

The first factor to examine is the computation time; fig. 2 shows only the computation speedup for STRIKE application (on the *y*-axis) for increasing number of nodes (on the *x*-axis). As expected the computation time is decreased in a semi-linear manner by increasing the number of nodes.

Next factor to examine is the communication effect. Fig. 3 shows the increase in communication time for increasing number of nodes as the number of messages required to be sent and received among nodes are increases.

TABLE III
Biblioteca Alexandrina Sun Microsystem Technical Description

Number of Nodes	128 eight-core compute nodes
Processors/node	2 quad-core sockets per node, each is Intel Quad Xeon E5440 @ 2.83GHz
Memory/node	8 GB memory per node, Total memory 1.05 TBytes (132 * 8GB)
Node-node interconnect	Ethernet & 4x SDR Infiniband network for MPI 4x SDR Infiniband network for I/O to the global Lustre filesystems
pre- and post-processing nodes	6 management nodes, incl. two batch nodes for job submission w. 64GB RAM
OS	OS, Compute Node: RedHat Enterprise Linux 5 (RHEL5) OS, Front End & Service Nodes: RedHat Enterprise Linux 5 (RHEL5)

Fig. 4 compares the relative communication time with the relative computation time for increasing number of nodes. The main observation is that the communication

overhead is negligible for lower number of nodes (i.e., 2 and 4 nodes) because it is less than 7%, but it cannot be neglected when using higher number of nodes (6 nodes) as it exceeds 12%.

VI. Conclusions

STRIKE was shown to improve upon the existing state-of-the-art methods for Protein-protein interaction prediction. We described the parallelization of STRIKE and its MPI parallel implementation and performance enhancement, specific algorithm enhancements and compiler flag enhancements, on a heterogeneous cluster system. On small protein sequence sets, the execution time of a parallel implementation of this bioinformatics algorithm was reduced to about 6 times when increasing number of nodes from one compute node to 5 compute nodes. PC cluster with 6 nodes takes a few communication time and scales the computation time near to linear. A higher number of nodes will improve computation performance if we increased the size of the protein sequences, but this will also effect on the communication cost. Our implementation was shown to scale very well with increasing data size and number of nodes.

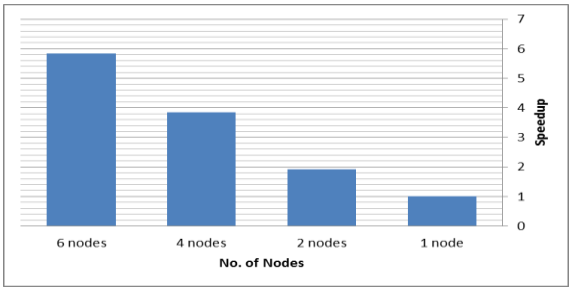

Fig. 2 Computation speedup vs. number of nodes using short-sequence set on small-scale cluster

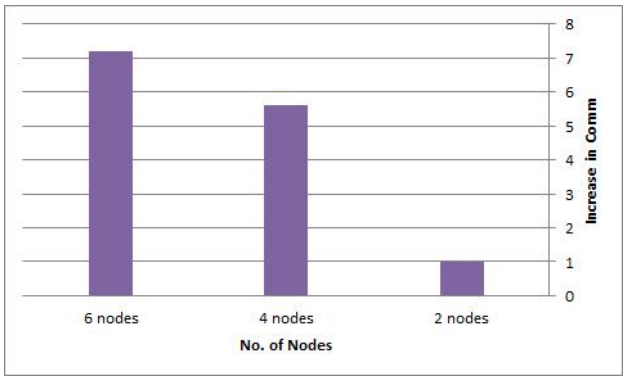

Fig. 3 Increase in communication time vs. number of nodes using short-sequence set on small-scale cluster

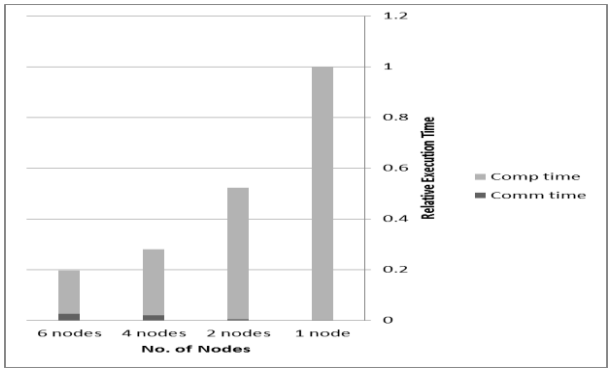

Fig. 4 Application performance using short-sequence set w.r.t 1 node on small-scale cluster

References

[1] Sprinzak,E. and Margalit,H. Correlated sequence-signatures as markers of protein-protein interaction. J. Mol Biol., 311, 2001, pp. 681–692.

[2] Deng,M. Mehta,S. Sun,F. Cheng,T. Inferring domain-domain interactions from protein-protein interactions. *Genome Res*., 12, 2002, pp. 1540-1548.

[3] Huang,T.W. Tien,A.C. Huang,W.S. Lee,Y.C. Peng,C.L. Tseng,H.H. Kao,C.Y. Huang, C.Y. POINT: a database for the prediction of protein-protein interactions based on the orthologous interactome. *Bioinformatics*, 20, 2004, pp. 3273-3276.

[4] Xue-Wen,C. Mei,L. Prediction of protein–protein interactions using random decision forest framework. *Bioinformatics*, 21, 2005, pp. 4394–4400.

[5] Sylvain,P. Frank,D. Albert,C. Jim,C. Alex,D. Andrew,E. Marinella,G. Jack,G. Mathew,J. Nevan,K. Xuemei,L. Ashkan,G. PIPE: a proteinprotein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs. *BMC Bioinformatics*, 7, 2006, pp. 365.

[6] Lodhi,H. Saunders,C. Shawe-Taylor,J. Cristianini,N. Watkins,C. Text Classification using String Kernels. *J. of Machine Learning Res*., 2, 2002, pp. 419-444.

[7] Zaki,N.M. Deris,S. Illias,R.M. Application of string kernels in protein sequence classification. *Applied Bioinformatics*, 4, 2005, pp. 45-52.

[8] Haussler, D. Convolution kernels on discrete structures. Technical Report UCSC-CRL- 99-10, University of California Santa Cruz, 1999.

[9] Watkins, C. Dynamic alignment kernels. Advances in Large Margin *Classifiers*, Cambridge, MA, MIT Press, 2000, pp. 39-50.

[10] Koonin, E.V. and Galperin, M.Y. Sequence-Evolution-Function: Computational Approaches, in *Comparative Genomics*, 2002, Kluwer Academic Publishers.

[11] Zaki,N. Lazarova-Molnar,S. El-Hajj,W. Campbell, P. Proteinprotein interaction based on pairwise similarity, *BMC Bioinformatics*, 2009, pp. 10-150.

[12] Zaki,N.M. Protein-Protein Interaction Prediction Using Homology and Inter-domain Linker Region Information, Lecture Notes in Electrical Engineering, Springer, 39, 2009, pp. 635-645.

[13] Sibai, F.N. ; Zaki, N., "Parallel protein sequence matching on multicore computers", Soft Computing and Pattern Recognition (SoCPaR), 2010.

[14] http://www.bibalex.org/ISIS/Frontend/Projects/ProjectDetails.aspx