PARALLELIZATION OF FUNCTIONAL FLOW TO PREDICT PROTEIN FUNCTIONS

A THESIS SUBMITED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRAH AKKOYUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MEDICAL INFORMATICS

JANUARY 2011

Approval of the Graduate School of Informatics

_____

Prof. Dr. Nazife BAYKAL

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assist. Prof. Dr. Didem GÖKÇAY

Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assist. Prof. Dr. Tolga CAN

Supervisor

Examining Committee Members

Assist. Prof. Dr. Didem GÖKÇAY (METU, II) _____

Assist. Prof. Dr. Tolga CAN (METU, CENG) _____

Assist. Prof. Dr. Özlen KONU (Bilkent U, BIO) _____

Assist. Prof. Dr. Yeşim Aydın SON (METU,II) _____

Dr. Cevat Şener (METU, CENG) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this wok.

Name, Last name : Emrah, Akkoyun

Signature : _____

ABSTRACT

PARALLELIZATION OF FUNCTIONAL FLOW TO PREDICT PROTEIN FUNCTIONS

Emrah Akkoyun

M.Sc., Department of Medical Informatics

Supervisor: Assistant. Prof. Dr. Tolga Can

January 2011, 51 pages

Protein-protein interaction networks provide important information about what the biological function of proteins whose roles are unknown might be in a cell. These interaction networks were analyzed by a variety of approaches by running them on a single computer and the roles of the proteins identified were used to predict the function of the proteins unidentified. The functional flow is an approach that takes the network connectivity, distance effect, topology of the network with local and global views into account. With these advantages, that the functional flow produces more accurate results on the prediction of protein functions was presented by the previos conducted researches. However, the application implemented for this

approach could not be practically applied on the large and complex network produced for the complex species because of memory limitation. The purpose of this thesis is to provide a new application be implemented on the high computing performance where the application can be scaled on the large data sets. Therefore, Hadoop, one of the open source map/reduce environments, was installed on 18 hosts each of which has eight cores.

Method; the first map/reduce job distributes the protein interaction network as a format which allows parallel distributed computing to all the worker nodes, the other map/reduce job generates flows for each known protein function and the role of the proteins unidentified are predicted by accumulating all of these generated flows. It has been observed in the experiments we performed that the application requiring high performance computing can be decomposed into worker nodes efficiently and the application can provide better performance as the resources increase.

Keywords: protein-protein interactions, functional flow, parallel and distributed computing, map/reduce, Hadoop

ÖZ

PROTEİN FONKSİYON TAHMİNLEMESİ İÇİN FONKSİYONEL AKIŞ YÖNTEMİNİN PARALELLEŞTİRİLMESİ

Emrah Akkoyun

Yüksek Lisans, Sağlık Bilişimi

Tez Yöneticisi: Yard. Doç. Tolga Can

Ocak 2011, 51 sayfa

Protein-protein etkileşim ağları, işlevleri bilinmeyen proteinlerin bir hücrede biyolojik fonksiyonlarının ne olabileceği ile ilgili önemli bilgiler sağlarlar. Bu etkileşim ağları, çeşitli yaklaşımların tek bir bilgisayar üzerinde koşturulması ile analiz edilmiş ve işlevleri bilinen proteinlerden bilinmeyenler tahmin edilmeye çalışılmıştır. Fonksiyonel akış; ağ bağlantısallığını, uzaklık etkisini, yerel ve global ağ topolojisini hesaba alma avantajına sahip bir yaklaşımdır. Bu avantajlarıyla protein fonksiyonlarının tahminlenmesinde daha başarılı sonuçlar ürettiği bundan önceki çalışmalarda gösterilmiştir. Ancak, bu yaklaşım için gerçekleştirilen

uygulama, gelişmiş canlılar için üretilmekte olan karmaşık ve büyük etkileşim ağları üzerinde bellek yetersizliği nedeniyle pratikte uygulanamamaktadır. Bu tez çalışmasındaki amacımız, Fonksiyonel akış yaklaşımının yüksek başarımlı hesaplama kümesi üzerinde gerçekleştireceğimiz yeni uygulama ile büyük veri setleri üzerinde ölçeklenebilirliğini sağlamaktır. Bu nedenle her biri sekiz çekirdekten oluşan 18 makine üzerine açık kaynak kodlu bir eşle/indirge ortamı olan Hadoop kurulmuştur.

Yöntem; ilk eşle/indirge işiyle protein etkileşim ağı, paralel dağıtık hesaplamaya izin verecek formatta tüm hesaplama uçlarına dağıtılır, bir başka eşle/indirge işiyle fonksiyonu bilinen her bir protein için akış üretilir ve üretilen tüm bu akışlar biriktirilerek bilinmeyen herbir protein için fonksiyon tahminlemesinde bulunulur. Yaptığımız deneylerde yüksek hesaplama gerektiren uygulamanın herbir hesaplama ucuna etkili bir şekilde dağıtıldığı ve kaynakların artırılmasıyla uygulamanın yüksek başarımla çalıştığı gözlemlenmiştir.

Anahtar Kelimeler: protein-protein etkileşimleri, fonksiyonel akış, paralel ve dağıtık hesaplama, eşle/indirge, Hadoop

To my family

ACKNOWLEDGEMENTS

First and foremost, I am heartily thankful to my supervisor, Assist.Prof. Dr. Tolga Can, whose encouragement, valuable guidance, friendly attitude and continuous support throughout my research enabled me to complete my study.

I also thank Assist.Prof. Dr. Didem Gökçay and Assist.Prof. Dr. Tuğba Taşkaya Temizel for their guidance and suggestions.

I would like to show my gratitude to my family for their love, patience and emotional support throughout my life. Without them, this thesis would never be finished.

I would like to thank all my colleagues and friends, especially to Onur Temizsoylu and Aydın Emre Ceviz, who always encouraged me with their understanding and valuable comments.

I am greatly appreciative to Ali Kantar, Ayse Ceylan and Sibel Gülnar in the institute for their kindness since the beginning of my M.Sc. study.

Finally, it is a pleasure to thank TUBITAK ULAKBIM, High Performance and Grid Computing Center (TR-Grid e-Infrastructure) where the numerical calculations reported in this paper were performed.

This thesis work is conducted as a part of TUBITAK Career Project # 106E128.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PPI	Protein-Protein Interaction Network
HPC	High Performance Computing
GFS	Google File System
HDFS	Hadoop Distributed File System
GO	Gene Ontology
PC	Personal Computer
SMP	Symmetric Multi-Processor
UMA	Uniform Memory Access
NUMA	Non-Uniform Memory Access
WLCG	Worldwide LHC Computing Grid
DIP	Database of Interacting Proteins
IB	Infiniband
QDR	Quad Date Rate

CHAPTER 1

INTRODUCTION

Firstly, the definition and motivation of the problem was given. By investigating the related works, the weakness and powerful points of the previously conducted researches were examined and what our contributions are indicated clearly in this chapter.

1.1 Problem Definition and Motivation

A protein cannot accomplish its biological function when it is completely isolated. Rather, it usually interacts with other proteins in order to perform vital part of the biological function such as cell growth, rRNA and tRNA synthesis, transcriptional control and cell polarity. The structure of the interactions with the proteins can be abstracted by a graph which is called the protein-protein interaction (PPI) network.

There is a variety of protein interaction networks which are obtained by various experimental techniques. They are generated for different species; as a result the number of proteins and interaction between the proteins are various. However, these datasets have high noise which means the linkage of the protein shown by the

network may not really exist biologically. Furthermore, most of the proteins in many PPI networks are not associated with any biological functions. The species which has most biologically identified proteins is the baker's yeast (Saccharomyces cerevisiae) although about one-fourth of the proteins remain uncharacterized. The biological techniques are not sufficient to associate all proteins with a biological function in a cell, therefore a number of computational techniques have been applied to understand proteins whose tasks are not known well [1,2,3,4,5]. However, noisy and incomplete PPI datasets makes the problem of function prediction using PPI networks a challenge.

There are many methods which are applied to predict function from a protein interaction network, and these grouped into categories based on neighborhood, global optimization, clustering and association by Pandey *et al* [1]. Functional flow by Nabieva *et al* [6] is one of these methods that takes the network connectivity, distance effect, topology of the network with local and global views into account, leading to significant advantages for more accurate function prediction.

The Functional flow is based on a well known network flow where the annotated proteins behave as source nodes, while the unannotated ones behave as destination nodes and the flow is propagated by the source proteins to the destination proteins throughout the edges like a conduit over the discrete of time. The amount of flow that enters the destination protein determines the biological function of the unannotated proteins. The computational time required for annotating proteins will be increased sharply, when the time steps and the number of the interactions or biologically known functions increase, which is a typical scenario especially for genome-scale networks. Therefore, the prediction of all the unannotated proteins in genome-scale protein interaction networks requires very long time when it is running on a single computer.

Hadoop is an open source project that enables users to run their application on a reliable, scalable and distributed computing environment with thousand of nodes and petabytes of data. It was inspired by Google's Map Reduce and Google File System (GFS) papers and used by many organizations to run large distributed computations. It is a kind of map & reduce programming model that rapidly processes vast amount of data in a parallel way on large clusters of compute nodes.

There are a few methods to predict protein function in a parallel way by utilizing the high performance computing technology. These methods are based on the protein sequence alignment that studies on the primary structure of a single protein or a small complex. In this thesis, we implemented a novel application on a number of powerful computers which are arranged by running together for protein function prediction by using interactomes. The functional flow was applied on the dataset and focused on the computational performance provided by the Hadoop platform instead of investigating the prediction results. The reason for this is that we applied a well known algorithm (Functional Flow) by previous researchers which provides accurate predictions for unannotated proteins. In order to evaluate the performance of Hadoop cluster, we installed it on 18 hosts with 144 cores which have the same operating system (Scienific Linux 4.7 - Beryllium) and middleware (hadoop_0.20-2). We run our parallel implementation of Functional Flow with various numbers of cores on the cluster. By examining the results with the distributed monitoring system for high-performance computing, it has been shown that significant performance has been gained. We also showed that all the resources installed on the cluster have been utilized by Hadoop platform very well. We examined that the Hadoop platform can facilitate bioinformatics studies which require high performance computation.

1.2. Related Work

Previously conducted researches which are related to our study were grouped by the computational techniques to predict protein function and Hadoop as a high performance computing platform.

1.2.1. Computational Techniques for Protein Function Prediction

There are several approaches that attempt to predict protein function from a protein interaction network. Majority method proposed by Schwikowski *et al.*[2], which is one of the neighborhood-based approaches, looks at the neighboring interactions and takes the three most frequent annotations. This is a basic method which can be applied easily. However, it is not good at prediction when there are quite a lot of unannotated neighbors, for the majority method only promotes the immediate

neighbors within any sub-network. Hishigaki *et al.*[3] extends the Majority rule by looking at all proteins within an area bordered by a certain radius. It annotates proteins with a function which is mostly found in the set that includes the frequencies of all the function inside a particular area. It takes advantage of the underlying network structure as well as the interactions beyond the immediate linkage; however, it is insufficient to promote any aspect of network topology. While it considers the number of functional annotations, it does not consider the linkage within the local neighborhood. Vazquez *et al.*[4] and Karaoz *et al.*[5] proposed a new method that exploits the global topological structure of the interaction network. It is a kind of well-known multiway k-cut algorithm that tries to cluster the interaction network by taking dense regions in the network into account. Then, proteins are labeled by one of the functions within the module. The size of the module might be large and a few numbers of annotated proteins might exist, and as a result, all the proteins might be labeled as one of these functions. This might cause incorrect function predictions. Nabieva *et al.* [6] proposed a method based on a well-known network flow, which is mostly applied for the graph cutting problem. There will be a flow from source nodes to the destination nodes over the edges in the PPI graph. It promotes both the underlying topology of the graph and multiple edge-disjoint interaction paths between two proteins. Furthermore, it takes the network topology into account. Thus, an algorithm has been proposed to overcome the weakness of the previous methods. From these methods, numerous innovative methods can be obtained to predict functions from protein interaction networks in computer science. It is also expected to adapt new techniques in computer science, such as social network mining and web search.

1.2.2. Hadoop Map/Reduce Programming for High Performance Computing

Hadoop, which is a kind of open source framework written in Java, is mainly for large-scale distributed batch processing infrastructure which runs on commodity computers [14]. The main advantage of Hadoop is its ability to scale to hundreds or thousands of nodes in a cluster. Furthermore, it can handle vast amount of data efficiently over a set of computers.

Hadoop Distributed File System (HDFS) splits large amounts of data into many smaller parts which are distributed separately across multiple nodes. It is comprised of a name node, which stores all metadata such as file name, permissions and the distribution information, and data nodes, which store the part of the data called as chunks. The name node coordinates the distribution of files, as well as monitors the possible failures which might occur on data nodes.

Hadoop is mostly appropriate for the applications with large data processing tasks such as searching and indexing, for it can distribute chunks of data to nodes in the cluster reliably and cheaply, and computation is done where data is stored.

A Hadoop-cluster might consist of thousands of nodes; therefore, it is highly probable that various troubles might occur. As a result, the framework should have a high degree of fault tolerance, detect the trouble automatically and fix it as soon as possible.

MapReduce is a commonly used paradigm which is available in many programming languages. Map basically applies a function to a set of elements and returns a set of results, whereas Reduce basically applies a function to a set of elements by considering the current result and the next element in the set. A map/reduce job is usually monitored by one JobTracker and a TaskTracker per data node.

Many organizations such as Yahoo, Amazon, Rackspace, Facebook and Wikia are using Hadoop for searching assist and data mining, searching index, session analytics and log processing.

1.2.3. Parallel Approaches for analyzing large-scale interactome

Due to fundamental physical limitations and power constraints on a single computer, the use of multicore algorithmic techniques is required to analyze large-scale protein-interaction networks in an efficient way. Although there are a variety of computational techniques to predict protein functions, none of them is implemented by utilizing the high performance computing technology.

There are a few computational methods to analyze large-scale interactomes for clustering them into sub-networks or inferring the evolution of proteins instead of directly predicting protein function. The first method proposed by Yang et al [22] analyzed the size of large interactome in order to cluster it in a parallel way. Their method is based on the edge betweenness clustering algorithm whose remarkable performance in discovering clustering structures in several networks was already showed by Girvan et al [23]. The clustering tool was written in C++ under Linux with the requirement of LAM software and Boost Graph Library and they achieved almost linear speed-up for up to 32 processors. However, clustering the interactome for protein function prediction is not suitable because it might cause incorrect function prediction, when the size of a sub-network is large and the sub-network has a few annotated proteins. The other method proposed by Bader et al [24] is the analysis of the degree–betweenness centrality correlation in the human protein interaction network to elucidate essentiality and evolutionary age of a protein. They designed a portable parallel implementation by using thread programming. The thread computing allows application to utilize all the processors on multi-core computers; however, utilizing more than a single computer is not possible. Therefore, the complex problem cannot scale on the number of computers running together to solve large problem.

In this study, we implemented a novel application on a number of powerful computers which are running together for protein function prediction. By using the Hadoop platform, the successful implementation of functional flow can be deployed on hundreds of computers without any change on the application.

1.3. Contributions

Our contributions in this thesis are:

1. The method of the functional flow for protein function prediction has been implemented by Java on a single computer. It is not possible to run the same implementation on Hadoop platform in a parallel way, therefore the method has been implemented again by taking MapReduce paradigm into account.

2. The complexity of the problem has been investigated in detail. To do that, several PPI networks with different sizes have been used as an input and the program has been run with various metrics to understand their effects on the running time well.

3. A new Hadoop cluster with 144 cores has been installed and tested by submitting a simple run which uses a number of cores concurrently. Furthermore, a tool for monitoring distributed computing cluster, which is commonly used by the High Performance Computing Centers, has been deployed and the performance provided by Hadoop has been shown from different perspectives.

4. It is important for the HPC platform to utilize the dedicated resources efficiently. How the dedicated resources have been facilitated has been represented with the graphs. Furthermore, the optimal configuration for Hadoop in order to gain the best performance has been shown.

5. For any PPI networks whose size are so large and/or complex that it is impractical or impossible to solve them on a single commodity computer because of limited computer memory, a new implementation on Hadoop has been proposed for solving the problem with multiple compute resources.

1.4. Thesis Outline

This thesis is organized as follows: In Chapter 2, we provide the necessary background knowledge to understand the problem domain and the solutions. In Chapter 3, datasets are described and technical details of the implementation are given. In Chapter 4, experimental results which demonstrate the performance of Hadoop are shown. In Chapter 5, the thesis is concluded with summary and future directions.

CHAPTER 2

BACKGROUND

The basic topics which help readers to understand well were covered in this chapter. Firstly, the biological backgrounds of the study were introduced, and then the algorithm which is applied to dataset was formulated and defined step by step. Furthermore, the characteristics of the parallel and distributed computing with their benefits were indicated. Finally, the fundamental services running on Hadoop platform were introduced.

2.1. Proteomics and Interactome

Proteomics can be defined as a large-scale study of proteins especially in terms of their structures and functions. Furthermore, whole set of molecular interactions in cells is called as *interactome* and identifying proteins interactions is one goal of the proteomics [7].

Molecular interactions can occur between molecules which belong to various biochemical families, such as proteins, nucleic acids, lipids and carbonbydrates. The interactome contain several thousands of binary interactions for a given species. However, none of them is presently completed and their sizes are still condensable.

The most complete interactome produced until now is for Budding yeast with 170,000 gene interactions and 54 million two-gene comparisons [8].

Sequencing whole genomes for organisms is not sufficient to predict what functions of proteins in the complex biological pathways of the cell are. After that, scientists start to investigate how proteins interact with their neighborhood to get a clue about their roles, and Proteomics emerged as a new field in Biology. Identifying the interactions between proteins is an important task for annotating functions. It might be useful to explore new biomarkers or to find new drugs for treatment of human diseases.

2.2. Protein-protein Interaction Network

A protein cannot accomplish its biological function when it is completely isolated. Rather, it usually interacts with other proteins in order to perform its function. A protein generally interacts with much more than one other protein and the structure of the interactions with the proteins can be abstracted by a graph which is called protein-protein interaction (PPI) network. The network of interactions between proteins is represented as an undirected graph G= (V, E) where V is the set of nodes indicating the proteins and E is the set of the edges indicating the interactions. If u and v are the proteins where $u,v \in V$ and there exists an edge between u and v, the corresponding proteins interact physically. Furthermore, the reliability of the edge are shown as a weight $w(u,v)$ where u and v are the interacting proteins. These are the significant terms that define any undirected and weighted graph.

Most of the proteins in existing PPI networks are not associated with any biological function. Figure 2.1 shows the rate of the proteins with unknown functions against to whole proteins in the network for different species [R. Sharan *et al*]. As it is shown, the species which have most biologically identified proteins are yeast (S.cerevisiae), although about one-fourth of the proteins remain uncharacterized.

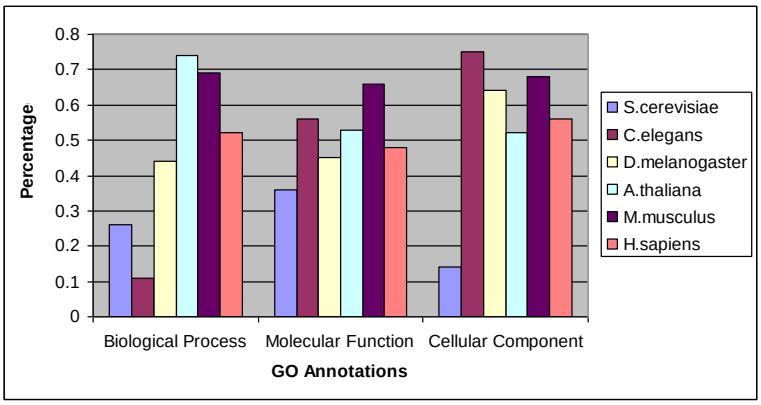


Figure 2.1 Fractions of the annotated proteins against to whole ones according to GO annotations

It is widely accepted by the researchers that it is highly possible the proteins which lie closer to one another in a PPI network have similar function. The following graph in Figure 2.2 shows the correlation between network distance and functional distance [Lord et al]. As the graph shows, the closer the proteins in the network are, the more similarities are observed regarding to biological function.

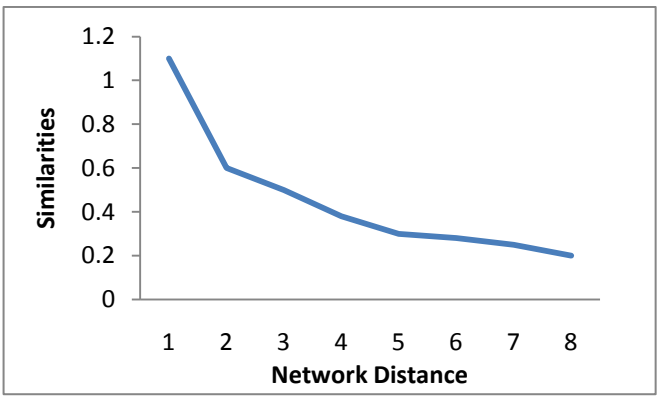


Figure 2.2 Correlation between protein functional distance and network distance. The proteins which are close to each other has more similarities than the proteins far from to each other.

A subgraph of a PPI network is given in Figure 2.3 [Vazques et al]. Demonstration of a simple method for predicting protein function is represented here as an example. Proteins in gray boxes are unannotated, while the proteins in other boxes are annotated and their associated functions are given in the brackets. The classified functions with their random label ids are given below in Table 2.1.

Table 2.1 Classified biological function with their label id

Label	Biological Function	Label	Biological Function
1	cell growth	7	tRNA synthesis
2	budding, cell polarity and filament formation	8	transcriptional control
3	pheromone response, mating-type determination, sex-specific proteins	9	other transcription activities
4	cell cycle checkpoint proteins	10	other pheromone response activities
5	Cytokinesis	11	stress response
6	rRNA synthesis	12	nuclear organization

In order to predict one of the unannotated proteins such as YNL127W, simply, it is required to have a look at the known functions which the surrounded annotated proteins have and the shared functions by the neighbors (function 2 for protein YNL127W in this example) can be basically assigned to the unannotated protein.

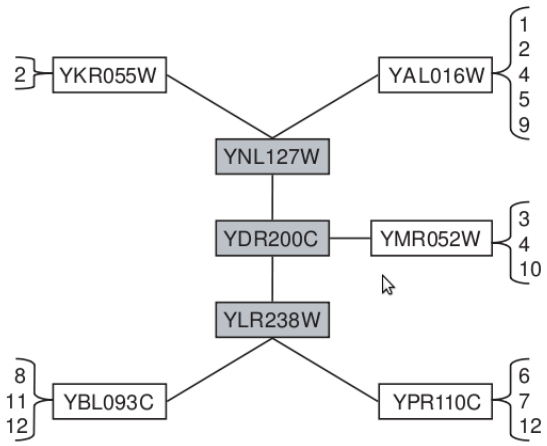


Figure 2.3 A subgraph of the protein interaction network of the yeast Saccharomyces cerevisiae. The simple technique to assign a protein function to unannotated proteins was demonstrated.

The following representation shows a larger set of interactions between the proteins for yeast [Wagner *et al*].

Figure 2.4 Visualization of protein interaction network in Yeast. The small circles show the proteins and the lines between the circles show interaction between these proteins.

2.3. Functional Flow

The algorithm [6] is based on well known network flow which is mostly applied for the purpose of graph cutting. Given a graph, network flow simulates a flow from source nodes to the destination nodes over the edges. In a PPI network, the annotated proteins behave as source nodes, while the unannotated proteins behave as destination nodes. The flow is propagated by the source proteins to the destination proteins throughout the edges like as conduit over the discrete of time. It is expected that the close proteins to the annotated proteins have more similarities than the proteins that are far from the source ones. Therefore, the effect of each annotated protein on any unannotated ones depends on the distance separating these two proteins and it is called as distance effect. Another constraint that the algorithm defines is the capacity of the edge. It determines the maximum amount of flow which can pass through the edge over time. Thus, it takes into account the network connectivity. If any unannotated protein has more paths to the source, it means that

the protein will obtain more flow. Furthermore, it prevents any protein to propagate too much flow to another protein in the network. After a fixed number of iterations, each protein has a flow which comes from various annotated proteins that finds out the functional score used to predict functions of unannotated proteins. If there is no flow that enters the protein, the functional score will be zero and it will not be associated with this function.

Reservoir defines the maximum amount of flow that a node can pass on to the neighbors, while the capacity defines the amount of flow that an edge can pass on. The annotated proteins have infinite reservoir that enables them to propagate flow to the surrounded proteins as long as it requires. At each iteration, the reservoir amounts of the proteins are updated. There will be always flows from proteins which have more filled reservoir to the proteins which have less filled reservoirs (downhill).

For each protein *u* in the interaction network, $R_0^a(u)$value represents the amount of reservoir for function *a*, at time 0 for protein *u*. The value of the reservoir is infinite, in case protein *u* is one of the annotated proteins.

$$R_0^a(u) = \begin{cases} \infty, \text{ if u is annotated with a} \\ 0, \text{ otherwise} \end{cases} \qquad \text{(Equation 1)}$$

g_t^a represents the amount of flow from the source protein *v* to destination protein *u*, or from source protein *u* to destination protein *v* at time *t* for function *a*. At initial step (*t=0*), there is no flow on the edges, therefore the *g* value is 0. The amount of reservoir is updated for each iteration or time step by calculating the amount of flow that both enters and leaves the nodes.

$$R_t^a(u) = R_{t-1}^a(u) + \sum_{v:(u,v)\in E} (g_t^a(v,u) - g_t^a(u,v)) \qquad \text{(Equation 2)}$$

At each iteration, the flow will proceed downhill and satisfy the capacity constraints. It cannot exceed the maximum capacity of the edges which have been already determined in the PPI network. Furthermore, the flow can be propagated only if the amount of reservoir that the source protein has higher than the amount of reservoir that the destination protein has.

$$g_t^a(u,v) = \begin{cases} 0, R_{t-1}^a(u) < R_{t-1}^a(v) \\ min(w_{u,v}, \frac{w_{u,v}}{\sum_{(u,y)\in E} w_{u,v}}), otherwise. \end{cases} \quad \text{(Equation 3)}$$

Finally, $f_a(u)$ value is calculated in order to predict functions of unannotated proteins. It defines the amount of flow that enters the protein *u* for function *a* over *d* iteration steps.

$$f_a(u) = \sum_{t=1}^{d} \sum_{v:(u,v)\in E} g_t^a(v,u) \quad \text{(Equation 4)}$$

2.4. Parallel Computing

Software has been traditionally implemented in serial computations where a problem is broken into a discrete series of instructions executed one after another on a single computer with one CPU. Only one of these instructions can be executed during the serial computation. On the other hand, parallel computing is to concurrently use multiple computing resources, such as a single computer with multiple CPUs and/or a number of computers connected by network, in order to solve a computational problem. A problem is broken into discrete set of instructions which are executed simultaneously on different numbers of CPUs. By executing multiple instructions at any moment time, the computation problem is solved in less time with multiple computing resources than with a single computer.

Beowulf, one of the projects begun at NASA, has opened the door for low-cost, high performance computing cluster built from commodity PCs which are arranged to work together to solve scientific problems with the use of Linux and Open Source software. By this project, new standards and tools have been developed to make parallel computing making easier for developers to build scalable and portable parallel computer applications. After that, many research institutes, universities and companies started to build their own computing clusters whose costs are kept low, while computing performance is increased.

The prominent components of a computer cluster are multiple computers (nodes), operating systems, fast network switches, network interface cards, fast

communication protocols and cluster middleware. All of them can be formed into a single system the users of which gain great benefits.

Parallel computing is used for modeling difficult scientific and engineering problems existing in the real world. Earth, environment, biotechnology, genetics, molecular sciences, seismology, circuit design and microelectronics are some examples. Furthermore, the applications, which require processing large amount of data, utilize parallel computing. Data mining, oil exploration, web search engines, medical imaging and diagnosis, financial and economic modeling are some examples of these applications. As given examples by Barney *et al* [6] show, a high number of disciplines are using parallel computing to solve large and complex scientific problems.

2.4.1. Terminology of Parallel Computing

There are numbers of terms associated with parallel computing should be covered in order to give a brief overview. The terms listed below are defined by Barney *et al* [6].

Task

"A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor."

Parallel Task

"A task that can be executed by multiple processors safely (yields correct results)"

Serial Execution

"Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. "

Parallel Execution

"Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time."

Symmetric Multi-Processor (SMP)

"Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing."

Communications

"Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network."

Synchronization

"The coordination of parallel tasks in real time, very often associated with communications. Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase."

Granularity

"In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.

Coarse: relatively large amounts of computational work are done between communication events

Fine: relatively small amounts of computational work are done between communication events"

Parallel Overhead

"The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:

- Task start-up time
- Synchronizations
- Data communications
- Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
- Task termination time"

Massively Parallel

"Refers to the hardware that comprises a given parallel system - having many processors. The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands."

Embarrassingly Parallel

"Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks."

Multi-core Processors

"Multiple processors (cores) on a single chip."

Cluster Computing

"Use of a combination of commodity units (processors, networks or SMPs) to build a parallel system."

Supercomputing / High Performance Computing

"Use of the world's fastest, largest machines to solve large problems."

2.4.2. Benefits of the Parallel Computing

Saves time and money: By arranging a number of computers to work together will shorten its time to completion. Furthermore, a cluster can be built from commodity computers with open-source software cheaply.

Solves larger problems: It is not possible to solve complex scientific problems which require large memory or processing huge amount of data on a single computer practically. Therefore, parallel computing is an obligation for these kinds of problems to solve.

Provides Concurrency: The parallel computing provides the concurrency by running discrete set of instructions on multiple computing resources simultaneously. The Access Grid (www.accessgrid.org) is a good example for a global collaboration network where people from around the world meet and conduct work.

Uses Computing Resources Effectively: A set of idle computers within a network can be used efficiently by assigning a number of tasks of any large problem.

Overcomes the Limitation of Serial Computing: The technology reached the physical limits of the processors design as regards to transmission speed, miniaturization and economic developments. The data cannot move faster than light speed through hardware and a certain number of transistors can be placed on a chip because of electronic limitations. Furthermore, it has become expensive to design a single processor faster.

The technological improvements on the network connectivity making the communication between nodes much faster and cheaper, as well as computers with multi-processors show that parallelism will become more important in the future.

2.4.3. Memory Architecture

The memory can be categorized with three groups such as shared memory, distributed memory and hybrid distributed-shared memory.

Shared Memory

A number of CPUs shared the same memory resources as global address space and only one processor can reach the shared memory location at any moment time. Therefore, synchronization is required for managing the tasks which are attempting to read and write the same location. It is categorized into two groups in terms of memory access time.

Uniform Memory Access (UMA): It is mostly found on SMP machines where access time to memory by all CPUs is equal.

Non-Uniform Memory Access (NUMA): Linking two or more SMP computers can be called as NUMA. A CPU can also access to memory which is located another SMP, but the access time is slower. Therefore, all the processors can reach the memory with various access times.

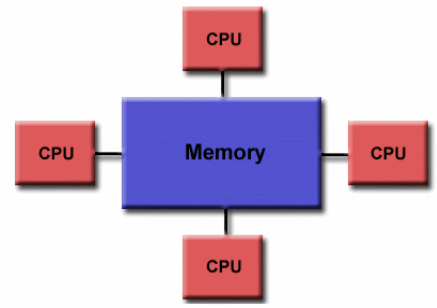


Figure 2.5 The representation of shared memory architecture (Uniform Memory Access)

There are two main advantages of shared memory computers. First, it is easy for developers to use memory efficiently. Second, data sharing among the tasks is fast due to speed of memory access. However, the cost of building these kinds of computers is high with increasing number of processors. Furthermore, synchronization is vital part of the parallel application in order to use global memory correctly.

Distributed Memory

The processors are operating independently, because of having their own memory. A change on the memory does not effect the other memory. There is a link among the inter-processor memory such as Ethernet, Infiniband for the tasks to share data by using message passing.

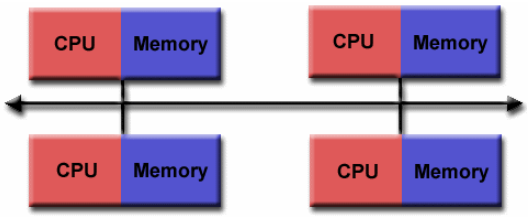


Figure 2.6 The representation of distributed memory architecture

Extending the computer resources with commodity, off-the-shelf processors and networking is less expensive and each processor can rapidly access their own memory without any interference. On the other hand, the data communication between CPUs is a challenge for the programmers.

Hybrid Distributed-Shared Memory

The processors on a SMP machine can access their memory as global. Furthermore, they can access the memory on another SMP by moving data through network communications established between the multiple SMPs. Thus, it takes both advantage of the shared and the distributed memory architecture.

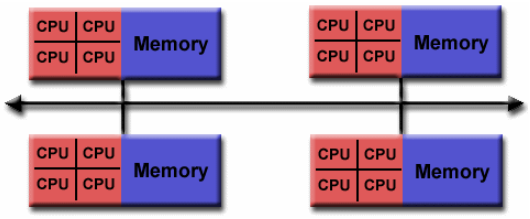


Figure 2.7 The representation of hybrid distributed-shared memory

2.4. Parallel and Distributed Computing with Hadoop

Hadoop is one of the parallel and distributed computing platforms which enables users to process vast amount of data in a parallel way. Any scientific problem, running on a single computer, could not be handled because of memory limitation or taking terribly long time to be solved. However, it can be figured out by Hadoop reliably and efficiently with using multiple computing resources simultaneously.

Hadoop is composed of two fundamental parts which are Hadoop Distributed File System (HDFS) and MapReduce Engine. HDFS is responsible for managing hundreds to thousands terabytes of data with high reliability, whereas MapReduce Engine is responsible for resource allocation and job management with automatic recovery of failure.

2.4.1. Hadoop Distributed File System (HDFS)

The file system consists of single *NameNode*, which records meta-data information of files, and a variety of *DataNodes* which store blocks of data. The large amount of data that has been split into many blocks, whose default size is 64MB, are replicated on three different data nodes. If one of them goes DOWN, blocks of data, stored on

the down node, can be recovered with accessing the replicated data which has already stored on the other data node. Thus, Hadoop platform can handle failure of data nodes in the cluster. On the other hand, it could not run an application whenever the name node, which is single, was down (single point of failure). Because of this reason, it is important to take snapshots of the name node's directory information in order to recover any failure without replaying the entire journal of file systems which might take long time. *Secondary node* is responsible for this process on the cluster. It regularly connects the name node and backs up to directory information. A system admin can restart the name node by using checkpointed images which are produced by the secondary node. Figure 2.8 shows the general view of a file system in Hadoop.

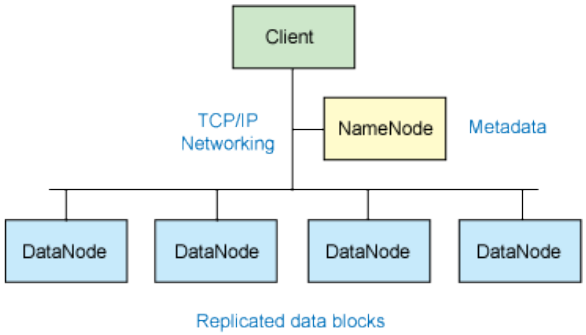


Figure 2.8 The view of fundamental services running on the file system in Hadoop

It is a great advantage for assigning tasks to the nodes on which the processed data has already stored, instead of copying data to where the tasks will be executed. It prevents high usage of internal network which might cause congestion on data transfer between data nodes. Furthermore, Hadoop takes internal network traffic into account regarding to location of nodes during automatic recovery of failure process. Initially, the node, which is located into same rack with the down one, is used to recover the lost or corrupted file, due to the fact that speed of transfers between the nodes on the same rack is much faster than the nodes on located on different racks. The other node on different rack will be used, in case both of the nodes on the same rack are down.

2.4.2. MapReduce Engine

There are two fundamental components of MapReduce Engine. The first one is a single *JobTracker* which manages the discrete of tasks on multiple computing resources in a cluster. The second one is a number of *TaskTrackers* which are running on each computing node in order to execute its own tasks. *JobTracker* is a vital part of the cluster, because the cluster goes down, when it fails. However, Hadoop can handle a variety of *TaskTrackers* failure with scheduling the failed work to the other *TaskTracker*. If it is a rack aware system, the *JobTracker* prefers the *TaskTracker,* located in the same rack, to execute failed tasks in order to reduce network traffic inside a cluster.

Any application, running on Hadoop platform, contains at least three pieces; *a map function* which takes a set of data and generates key/value pairs during the execution, *a reduce function* which obtains the generated key/value pairs and reduces the list of pairs in terms of their key and *a main function* which is responsible for job control and input/output file.

Developers need to make their code running on Hadoop platform, and then submit applications to the cluster. Firstly, *JobTracker* obtains information of input and output directories which is provided by *HDFS*. The number of tasks which will be executed on *TaskTracker* is determined by *JobTracker* with using knowledge about blocks of file. After that the hadoop application as well as its related part of file blocks is copied to every computing node. Finally, the tasks are started to process their blocks of file and reports status of the progress to the *JobTracker*. Figure 2.9 shows a work flow on Hadoop.

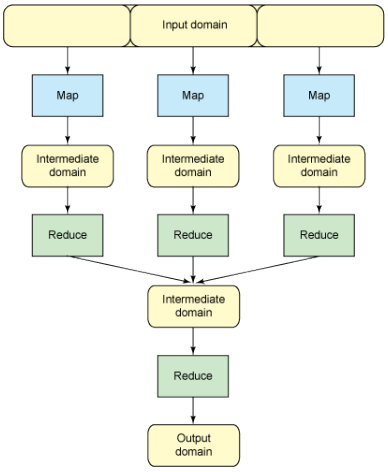


Figure 2.9 A demonstration the work flow of a process on Hadoop

Data communication between computing nodes (internal network traffic) affects the performance of parallel computing in a cluster. In this regard, Hadoop platform gives importance to keep amount of storage which will be moved low in order to gain high computing performance. To do that, it moves the processing to the storage, instead of moving storage to the processing.

2.5. Hadoop Computing Cluster and Architecture Employed in this Thesis Work

In any computing cluster, it is important for worker nodes to have same architecture and operating system, as well as middleware. Therefore, the Hadoop cluster has been installed on 18 hosts with 144 cores which have the same operating system (*Scienific Linux 4.7 - Beryllium*) and *hadoop 0.20-2,* which is the latest version as a middleware. Scientific Linux is one of the Redhat releases that is widely installed on the sites all over the world. Nearly all the computing centers under WLCG (Worldwide LHC Computing Grid) use Scientific Linux as an operating system. WLCG is an e-infrastructure for grid computing that provides a great number of computing clusters with various sizes in order to satisfy the scientists' requirements.

Each worker node has *Quad-Core AMD Opteron Processor - 2356* with 2.3Ghz and 16 GB memory. This is a kind of multi-processors products where a number of processors (cores) are placed on a single chip. The hosts are connected to each other with ethernet technology. A node is assigned to computing element which runs the

NameNode and *JobTracker* services for managing the tasks and distributed data over the cluster. This is called *master*, while the other nodes are called *slaves* which run the *DataNode* and *TaskTracker* service to manage their individual tasks.

2.6. Monitoring Tool for HPC Cluster - Ganglia

Ganglia is just one of the widely used distributed monitoring systems for high-performance computing, such as cluster and grid computing. In order to evaluate the performance of the Hadoop cluster well, Ganglia is beneficial to monitor the CPU usage of the all worker nodes on the cluster. It gives a web-based result which is a user friendly environment with useful graphs to show historical usage of the resources. Ganglia has been deployed on a variety of the HPC sites all over the world. It is also available for the Hadoop cluster.

TR-03-METU is one of the computing centers which belongs to TR-Grid (High Performance and Grid Computing Center) e-infrastructure. It supports the CERN experiment as Tier-2 center which has a lot of jobs submitted daily by high energy physics users for their monte-carlo production. As a result, nearly all the resources are fully utilized by the physics community. Therefore, TR-03-METU is selected to describe the facilities provided by Ganglia. The following snapshots represent the web-based result for the resource usage for a week at METU.

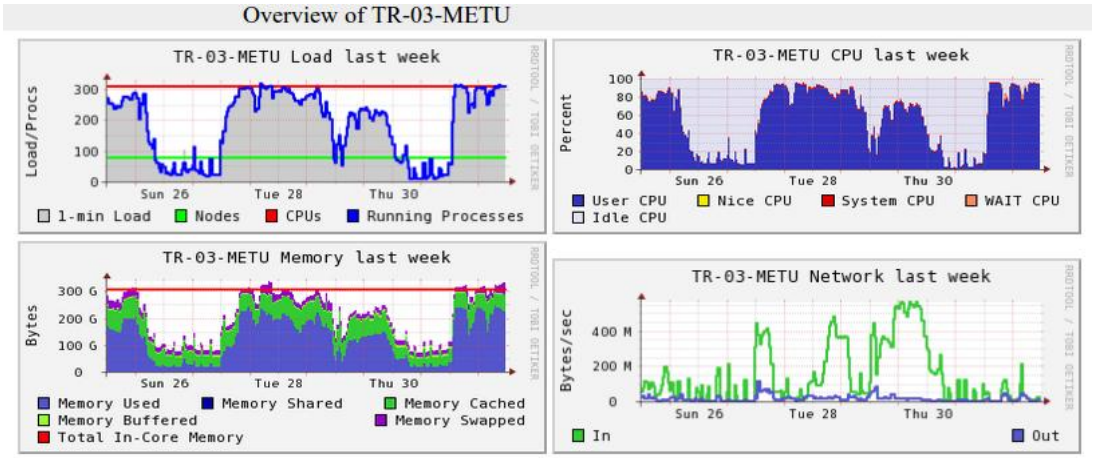


Figure 2.10 The average resource usage for a week at Tier-2 center. The number of running jobs, the total CPU and memory load and internal network traffic is represented in real time by Ganglia.

Ganglia shows the number of worker nodes with their status and the total number of cores in real time. According to this result, there are 77 worker nodes available and all of them are UP. Thus, the available core number is totally 308, because each worker nodes has a processor with four cores. The average load of the site is also given by Ganglia periodically. The average load for last 15, 5 and 1 minutes is 99%, 100% and 100%. The graph shows past usage during the last week with various metrics.

The first one is related to number of running jobs on the time zone. The red line shows the number of maximum concurrent running processes, whereas the green line shows the number of the UP hosts. The green one is straight which indicates that no problem occurs on the cluster, because no hosts were DOWN during this particular week. The blue wavy line shows the number of simultaneously running jobs over the time. As it is shown, the cluster is fully populated with the submitted jobs on the last days of the week, whereas there are many idle nodes at the beginning of the week.

The second graph titled *TR-03-METU CPU last week* shows the CPU load by the users. The CPU usage by the system processes is very low according to CPU usage by the users. Furthermore, the wavy line is very similar to first graph, since the CPU load mostly depends on the running of jobs on the cluster.

The third graph titled *TR-03-METU Memory last week* shows the status of the memory during the previous week. It is very important for the system administrators to understand the requirements of the memory that the worker nodes should have. The characteristics of the submitted jobs to the cluster might be very different from each others; therefore, their memory requirements are various as well. If the memory is not sufficient to run jobs on the worker nodes, the swapped memory will be used, which causes a sharp performance loss. Therefore, the metric of memory swapped should be investigated and there should be memory update, if the usage of the swapped area is high. Furthermore, the graph gives the information about cache and buffered memory usage as well as sharing memory usage. The red line shows the total amount of the memory the cluster has.

The final graph titled *TR-03-METU Network last week* shows the internal network traffic inside the cluster. Tier-2 center has a central storage system that has a great

amount of disk space where all the worker nodes can reach. It is a kind of shared file system over the nodes in the cluster. The jobs submitted by physicists analyze about 1.5 GB file for MC production; therefore, it requires to copy file from storage element to worker node, which causes the internal network traffic. Any congestion or problem on the transfer can be detected. As it is shown, the higher the number of running processes, the higher the internal network traffic are.

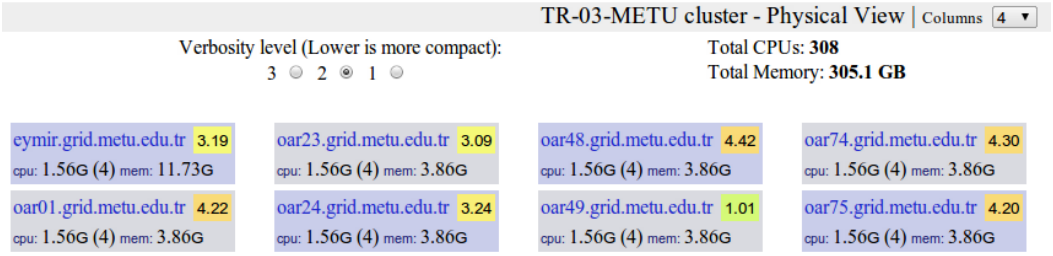


Figure 2.11 The physical views of the worker nodes. The properties of the computing nodes as well as the current usage of the resources such as CPU and hard disk were presented.

The historical usage of the worker node is also available on the Ganglia with fully view mode. However, the above figure shows the worker nodes at physical view. Firstly, it gives the total number of CPUs and available memory, as well as total amount of hard disk on the cluster. The usage of the disk on the node is an important metric, since the jobs fail when it is full. Therefore, Ganglia shows the node whose disk usage is the highest over the other nodes. Furthermore, it gives information about the worker nodes such as CPU architecture, the number of cores and memory. The most important part is the load value, because the hosts are expected to have a maximum of 4.0, since it has Intel architecture with 4 cores. If the load value is much higher than the number of cores, there is a misconfiguration on the site and the job could not be well partitioned over the cluster. It is a good metric to set optimum number of maps for each node in this study.

CHAPTER 3

MATERIALS AND METHODS

In this chapter, the implementation of the application utilizing the high computing performance provided by Hadoop was described in detail. Furthermore, the results of the application provided by Hadoop user interface were analyzed and the performance of Hadoop were monitored by Ganglia which is an external tool that provides information about historical usage of the dedicated resources.

3.1. Data Sets

There are a variety of protein-protein interaction networks for different species. These networks have high noise which means that some interactions between proteins might not really exist. Furthermore, the networks are not complete meaning that a lot of true interactions may not be present in the network. This situation prevents an accurate function prediction for unannotated proteins. Scientists use various techniques to estimate the reliability of the linkage between proteins. Investigating the amino acid sequence of proteins is just one of used methods. Furthermore, computational techniques such as data mining have also been used to determine the confidence of an interaction. This technique simply looks at a number of shared neighbors and assigns a value depending on the shared neighbors of both proteins [Chen e*t al.* 2007; Pandey *et al.* 2007]. Another example of these techniques

is to combine various sets of interactions obtained by experimental techniques, such as the two-hybrid method, the protein chip method and the tandem affinity purification method. As a result, a network associated with confidence value for each interaction is generated and it is called a *weighted protein-protein interaction network.*

3.1.1 Weighted Protein-Protein Interaction Network

In this study, a weighted PPI network where the weight values on the edges are initialized with 1.0 is used. The used PPI network was compiled by DIP [26] database on 4 Nov 2007. The total number of proteins is 9224 and the number of interactions between these proteins is 17491. The weight value has a significant impact on the prediction results, but there is no effect on the computing performance. Thus, all the weights on the edges are assumed to be 1.0. However, it can take any single precision number. Instead of investigating the biological accuracy of prediction results, this study focuses on the computational performance provided by the Hadoop platform, because a well known algorithm (Functional Flow) [Nabieva *et al.* 2005] has already been applied by previous researchers, which provides accurate predictions for unannotated proteins. Furthermore, the number of proteins annotated is 1281 according to the annotation data that was obtained for yeast from geneontology.org. The annotated proteins mean that their biological functions are already known.

The binary interactions between proteins as well as their weight information are stored in a text based file where each line corresponds to a linkage, while annotated proteins and their associated function IDs are stored in another text based file where each line defines a biological function and lists proteins that have this biological function. The contents of the input files are shown in the following tables (Table 3.1.1.1 and Table 3.1.1.2).

Table 3.1.1.1 Format of the text-based file representing all interactions

Protein A ID	Protein B ID
YMR056C	YBR217W
YMR056C	YJL124C

Table 3.1.1.2 Format of the text-based file representing a list of known functions

Biological Function ID	Protein A ID	Protein B ID	Protein C ID	Protein D ID	…
51219	YNL229C	YDR130C			
32184	YHR079C	YLL001W	YER148W	YDR054C	
8017	YDR285W				

3.1.2 Gene Ontology

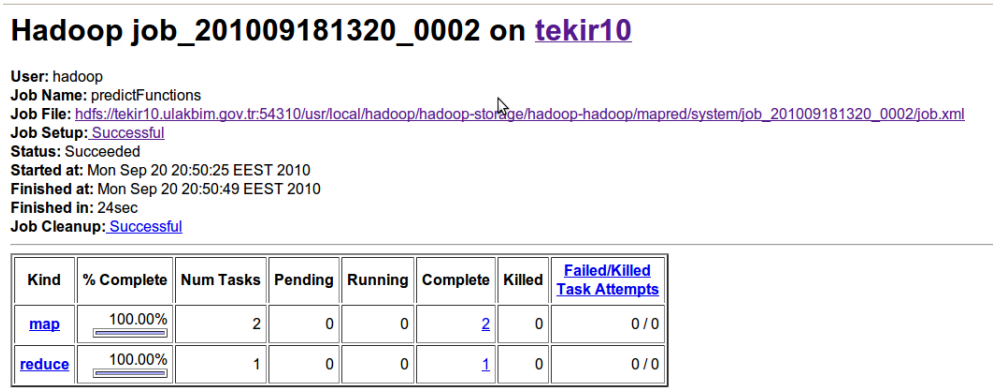
It is very important to unify the representation of genes and gene product attributes across all the species in order to find any functionally equivalent terms easily. For example, if one database describes a set of molecules as a translation, whereas another describes it as a protein synthesis, it will be difficult for people and computers to search molecules associated with the similar functional term. Thus, Gene Ontology (GO) becomes a bioinformatics initiative providing consistent descriptions of gene products in different databases.

The first aim of the Gene Ontology collaboration group is to maintain and further develop a generic vocabulary of gene and gene product attributes. The second aim is to annotate genes, and assimilate and disseminate annotation data. Another aim is to provide a number of useful tools using the dataset provided by GO.

In this thesis, the file of gene annotations with biological functions has been prepared by using the recent yeast GO annotation file. We use the "Molecular Function" subset of the three main GO hierarchies: "Molecular Function", "Subcellular Component", and "Biological Process".

3.2 User Interface of Hadoop for evaluating performance

There are two ways to understand the status of the jobs submitted to the Hadoop cluster. The first is to track the output which is produced by the Hadoop user interface on the console. It is a text based interaction system where users can manage their jobs. Another way to check the job status is a web based environment which is more user friendly. After a job has been submitted to the Hadoop cluster, both text and web based user interfaces basically show the current status of the job, the percentage of the task competition, total time for applications to be completed, information about the number of maps and reduces setting by the job configuration and the amount of data read and written by the HDFS. Figure 3.3.1 shows the information provided by the Hadoop user interface for an example job.



Hadoop job_201009181320_0002 on tekir10

User: hadoop
Job Name: predictFunctions
Job File: hdfs://tekir10.ulakbim.gov.tr:54310/usr/local/hadoop/hadoop-storage/hadoop-hadoop/mapred/system/job_201009181320_0002/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Mon Sep 20 20:50:25 EEST 2010
Finished at: Mon Sep 20 20:50:49 EEST 2010
Finished in: 24sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	2	0	0	2	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

Figure 3.3.1 The web based user interface of Hadoop gives information about the status of submitted jobs. Therefore, a user can monitor its own job.

The information about completed jobs can be classified under three main groups, such as *Job Counters*, *File System Counters* and *Map-Reduce Framework*. The number of reduce tasks as well as map tasks is important for evaluating the performance of the Hadoop cluster. The map and reduce number can be arranged according to the number of cores and the processor architecture the worker nodes have. Increasing the number of tasks means that the benefits of computing resources will be increased; consequently, the CPU running time for applications is expected to be decreased. This kind of information can be found under *Job Counters*. The other group is *File System Counter* which gives the information about bytes of file read and written by HDFS. Another group is *Map-Reduce Framework,* where given numbers might be a clue for determining the complexity of the calculations. It shows the number of records that map input and output operations have. The number of

records will be high, in case large amount of computation is required. Figure 3.3.2 shows an example web based user interface output.

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Launched map tasks	0	0	2
	Data-local map tasks	0	0	2
FileSystemCounters	FILE_BYTES_READ	0	106	106
	HDFS_BYTES_READ	19	0	19
	FILE_BYTES_WRITTEN	176	106	282
	HDFS_BYTES_WRITTEN	0	84	84
Map-Reduce Framework	Reduce input groups	0	8	8
	Combine output records	8	0	8
	Map input records	2	0	2
	Reduce shuffle bytes	0	112	112
	Reduce output records	0	8	8
	Spilled Records	8	8	16
	Map output bytes	84	0	84
	Map input bytes	12	0	12
	Map output records	8	0	8
	Combine input records	8	0	8
	Reduce input records	0	8	8

Figure 3.3.2 The web based user interface of Hadoop can provide information about the cluster configuration and application utilization

3.3 Debugging Hadoop Applications

As is the case with other computing platforms, debug operations on the implementation are not easy on the Hadoop computing platform either. There are a variety of maps that work concurrently and it is not possible to take a control over these individual operations. However, Hadoop provides developers to run their application on a single map. It means that a developer can run their implementation on any cluster by using only a map on a single core, and debug it. The standard output of the application can be reached by the console that gives developers a great convenience and control on their own application. For this purpose, the following parameter should be set in the job configuration class. In this study, therefore, the debug mode of the application was implemented as well.

```
conf.set("mapred.job.tracker", "local");
```

3.4 Overview

This section presents the overview of the algorithm used in this thesis. The major steps of the algorithm are as follows:

1. Transform input files provided by protein interaction database and gene ontology to a new text-based format that computers can process in a parallel way.

2. Generate a hash table which holds all interactions and distribute it to all computing nodes.

3. Start a number of processes concurrently to perform their own operations and generate key-value pairs where each one shows an individual flow by considering well defined formulas in the functional flow algorithm. Each process considers only one biological function and propagates a variety of flows assigned for that function.

4. Accumulate all the propagated flows and combine them in order to calculate the total amount of flows that enter an individual protein for each biological function in the network.

5. Compare the total amount of flows coming from each biological functions and annotate proteins with a function which has the highest value.

3.5 Pre-processing operations for Hadoop to propagate flows individually

A PPI network is represented as a graph which is composed of nodes and edges. For the parallel implementation, we use two input files for the set of interactions and functional annotations, as it has been stated in the previous sections. We preprocess one of the input files for the parallel implementation (details given below).

In the functional flow algorithm, each flow which is propagated from different annotated (source) proteins can be processed independently. It means that any protein u which is associated with function a and any protein v which is associated with

function *b* propagate their flows to the surrounding proteins connected individually. Different flows do not interact; as a result they can be processed separately. Each line of the input file is processed concurrently in a Hadoop computing platform. Therefore, all annotated proteins are categorized with respect to their function IDs. Unfortunately, the protein interaction database cannot provide the annotated proteins in that format. Therefore, it is required to process the input file in order to transform a new format that Hadoop can process in a parallel way. The preprocess operation is easily implemented. Therefore, the computing cost can be ignored compared to the total computation cost.

Figure 3.6.1 is a very simple demonstration of a PPI network. The annotated proteins are surrounded with red circles and numbers on the edges indicate weight values. The letters in the circle show the IDs of proteins and f1 and f2 values represent the function ID of the known proteins.

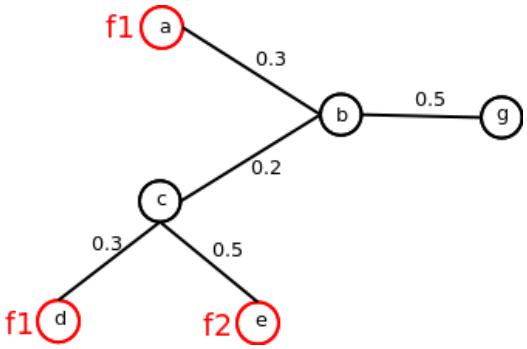


Figure 3.5.1 A simple example of a weighted protein-protein interaction network. The red circles show the annotated proteins, while the black circles show the unannotated proteins. The line shows the interaction with the weight value.

The text based representation of the graph is given below. The first one stores the set of interactions with weight values, while the second one stores annotated proteins with biological function ids. After processing, it has been transformed into a new format which Hadoop can process independently.

allInteraction.txt

a	b	0.3
b	c	0.2
c	d	0.3
c	e	0.5
b	g	0.5

annotatedProteins.txt

f1	a
f2	e
f1	d

3.6 Generating a hash table for a PPI network

In this implementation, two jobs have been submitted sequentially. The first one is a simple job which runs on a host and a single core is enough for it in order to complete its own task quickly. The task of this job is to generate a hash table which holds all the interactions as well as their weight information. It takes the file which contains the PPI network as an edge list as input and maps it on the memory with the appropriate hash table in an efficient way. It is also possible to complete this task by using many numbers of cores, but this will not speed application up, since the task is very simple.

In the map class, each edge of the PPI network has been processed one by one and generates key & value pairs for each line with weight information. Then, the reduce class collects all the key value pairs and generates the PPI network like as demonstrated in the following example.

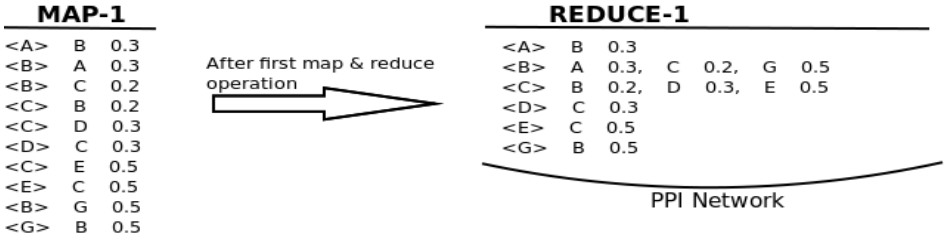


Figure 3.6.2 The demonstration of generating a hash table by using a file stores all the interactions between proteins in a PPI Network

The PPI network generated by the first job is important, because it stores all the interactions as well as their weight values. Whenever there is a flow propagated, this PPI network is used because the properties of linkages between proteins are defined here. The generated file which stores PPI network will be the input of the second job.

3.7 Mapping a hash table to all memories on the computing nodes

Each edge in the input PPI network is processed by the second job separately. Therefore, the number of maps should be defined in accordance with the number of edges. For each function, there will be flow propagated to surrounding proteins by

looking at the PPI network information. Thus, the PPI network should be published all over the computing nodes in the Hadoop cluster and mapped on their memory in order to increase computing efficiency. To distribute any files all over the computing nodes, the cache is used as follows:

```
DistributedCache.addCacheFile(new Path("/user/emrah/hash-output/part-00000").toUri(), conf2);
DistributedCache.createSymlink(conf2);
```

```
hdfs://tekir10.ulakbim.gov.tr:8020//user/emrah/hash-output/part-00000
```

In the functional flow algorithm, the total weight is mainly used to find out the amount of flow propagated. Hence, if the PPI network also stores the total weight information, this should increase the computing performance. During the map operation on the memory, the total weight information is also added to the hash table. As a result, it is not required to compute total weight information again and again for calculating all the amount of flow.

The amount of reservoir should be set as an infinite value, when the functions of the proteins already known are called source proteins. The annotated proteins can be seen as having infinite value of reservoir during all the updates. According to the formula of functional flow, the amount of reservoir of the annotated proteins will be decreased proportionally with respect to the amount of flow leaving from the source protein.

3.8 Propagation Flows in Parallel

The reservoir table holds the amount of reservoir that proteins have. There will be always a flow whenever the interacted protein pairs have reservoirs that are not the same. A flow will be propagated from a protein with higher amount of reservoir to a protein with less amount of reservoir. To detect whether there will be flow or not, the reservoir table is used for all the iterations.

The iteration number determines the number of time steps that the flow will be

propagated. When this number is set to a high value, it sharply increases the time the application takes to be completed. In order to understand Hadoop performance well, this number was set as high value in this study, although this may not be biologically meaningful. In the Functional Flow paper, this value has been set to 6, which is enough for a biological investigation.

For the initial step, the reservoir table is initialized with infinite values, because only annotated proteins exist in the hash table. After this, the flow is started to propagate until the time step reaches the iteration number the user has set. At each time step, the reservoir table is updated and the flow is propagated properly. The flows are collected into a result hash table within each map as follows:

<protein ID><function ID><amount of flow>

When the destination protein has a flow at any previous time step and there is a new flow, the value of the protein in the result hash table increases by the amount of the new flow appropriately. Thus, there is also a reduce operation inside a map in order to increase the computing performance.

Figure 3.6.4 is a demonstration of functional flow for function *F1*. The time step *t* shows the current iteration number, *Ra* shows the amount of reservoir that protein *a* has. *D => C, F1: 0.3* means that there is a flow which is propagated from protein *D* to protein *C* for function *F1* and the amount of flow value is *0.3* The reservoir of annotated proteins are *6000* and this has not changed. Since the portion of change is too small, it can be ignored.

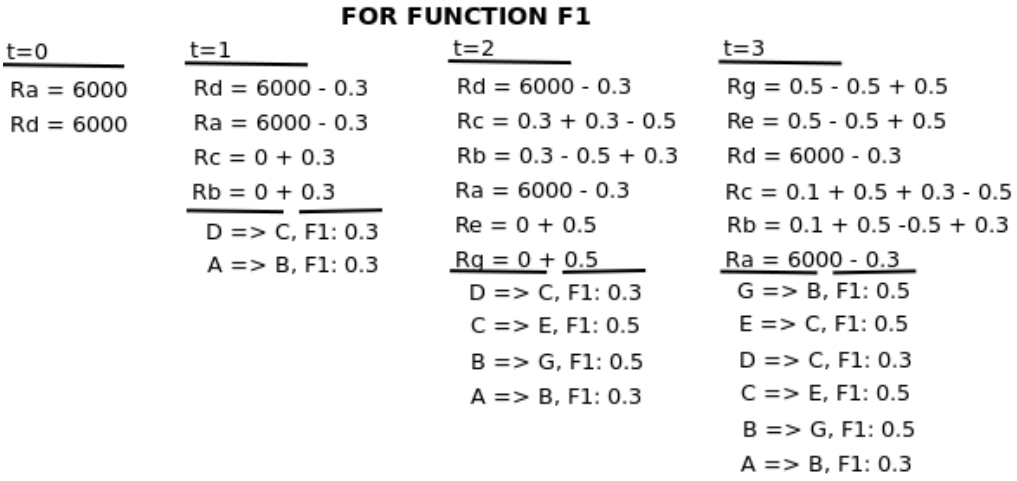


Figure 3.6.4 Propagating flows by regarding to previously given PPI network example. At each time step, there are a number of flows propagated for defined biological function id.

3.9 Accumulating all flows and making prediction

All the flows that each protein has from various functions are collected into a result hash table inside a map operation. However, there are also other flows collected inside other maps and all of them must be combined in order to predict function of the unannotated proteins. In order to collect all results produced by a number of maps, the reduce operation has been applied by Hadoop. With the help of Hadoop, the reduce operations are also done in a parallel way to increase the performance.

Table 3.6.5 shows the final results produced by the application on Hadoop for the given example. <B> F1, 1.4 shows that the protein B has 1.4 unit flows by function F1 in total. As the table indicates, protein B has more flow from function F1 than function F2. As a result, it will be annotated with function F1.

Table 3.6.5 Annotation of proteins with a biological function

Protein ID	Biological Function	Amount of Flow	Prediction of protein function
B	F1	1.4	
B	F2	0.4	B will be associated with function F1
C	F1	1.4	C will be associated with function F2
C	F2	1.5	
D	F2	0.6	D will be associated with function F2
E	F1	1.0	E will be associated with function F1
G	F1	1.0	G will be associated with function F1

Finally, all the unannotated proteins will be predicted to have one of the functions according to total amount of flow that protein has.

CHAPTER 4

RESULTS

In this chapter, we present the experimental results of our method applied on the yeast interactome whose properties are indicated in Chapter 3.

4.1 The Computational Complexity of the Problem

In this part of the study, some of the metrics for the application remain constant, while the others are changed in order to investigate the nature of the problem and the performance of the implementation as well as the utilizing the dedicated resources by Hadoop platform.

The application requires three parameters for assigning functions to the unannotated proteins in the network. The first parameter is a file which stores the information about the interactions between the proteins with the confidence weights. The second parameter is a file which stores the list of various biologically known functions for a given organism. The final one is a value which determines the number of time steps that the functional flow will be applied.

The complexity of the problem mostly depends on the number of iterations (time steps). It also depends on the amount of the interactions and the set of biologically known functions. However, they do not have a great impact on the running time as much as the number of iterations has. When the time steps is set to a high value, the running time required to annotate the proteins will be increased sharply, In order to clarify this statement well, the number of iterations has been changed, whereas the number of the interactions and the known functions remain the same (the input files are kept constant for all trials) and we analyze the value of *Reduce Input Records* provided by the Hadoop user interface. This record indicates the number of key & value pairs which holds an individual flow. High value shows that a great number of flows have been produced. As a result, the large amount of computation is required to apply the algorithm that is an indicator of the complexity of a problem.

Figure 4.1.1 demonstrates the effect of various iteration numbers on the number of flows. There is an increment on the flow numbers sharply, until the hop number is set to 14. Then, the increment of the flow numbers is nearly smooth, due to the fact that the input files are not sufficient for growing computational complexity of the problem, regardless of the high iteration numbers.

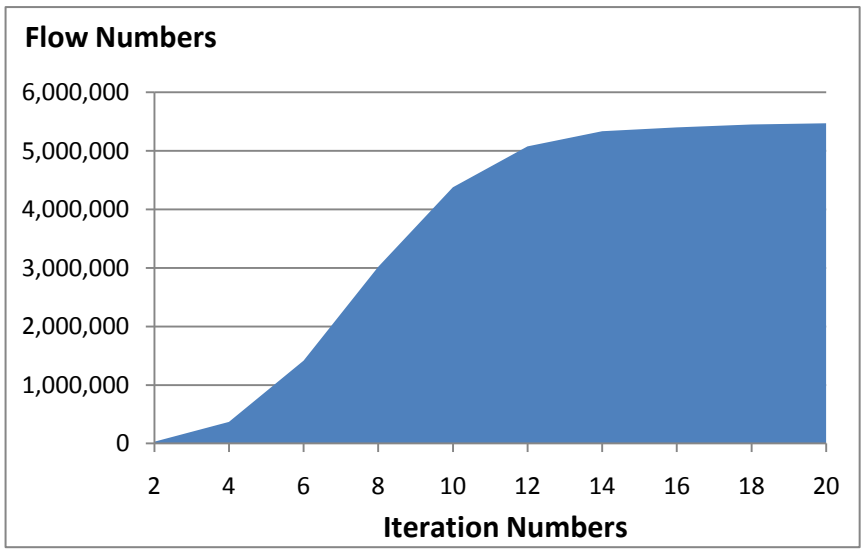


Figure 4.1.1 The change on the number of flows according to iteration numbers. The flow number shows the complexity of the problem and iteration number indicates the time step.

The time requiring the application to complete its task is continuously increased by the number of iterations, as it has been presented by Figure 4.1.2. Initially, there is no

great distinction on the increment of the running time, when the iterations number is set to a low value because the time for initializing of the jobs is relatively high according to the total running time. On the other hand, when the iterations number is set to a high value, the running time of the application is too high that the time for initializing the tasks can be ignored. The experimental results show that the number of iterations has a great effect on the flows number and the running time.

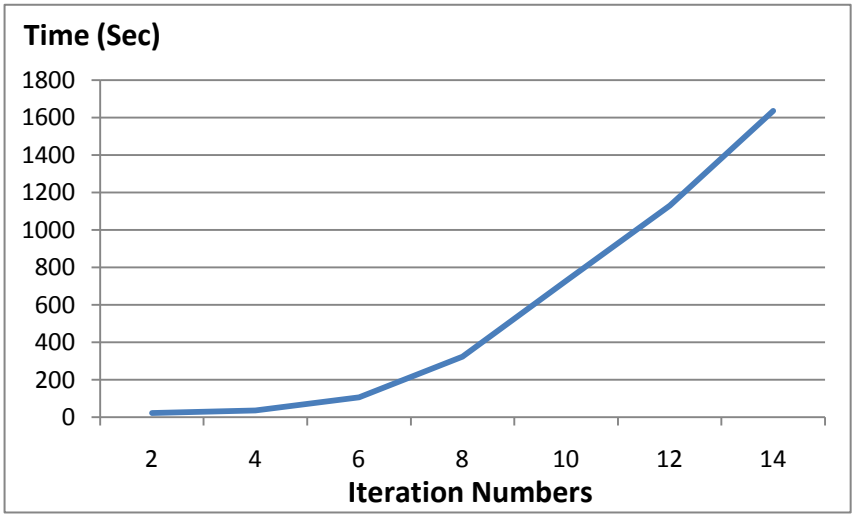


Figure 4.1.2 The change on the running time of the application according to the iteration numbers. As the iteration number increases, the time for application to be completed is increased as well.

4.2 The Evaluation of the Hadoop Performance by User Interface

Hadoop is one of the high performance computing (HPC) platforms where the running time of the applications are expected to be decreased in accordance with the number of cores installed into cluster is increased. Furthermore, it is expected to utilize all the dedicated resources fairly and precisely. Thus, each core should have a single map task at least. For example, a worker node with 8 cores should have 8 map tasks. In this part of the study, the Hadoop performance has been investigated with the same application under various number of the map tasks. Figure 4.2.1 shows the running time depending on the number of map tasks running on the cluster where each worker nodes has 8 cores and 16 GB shared memory.

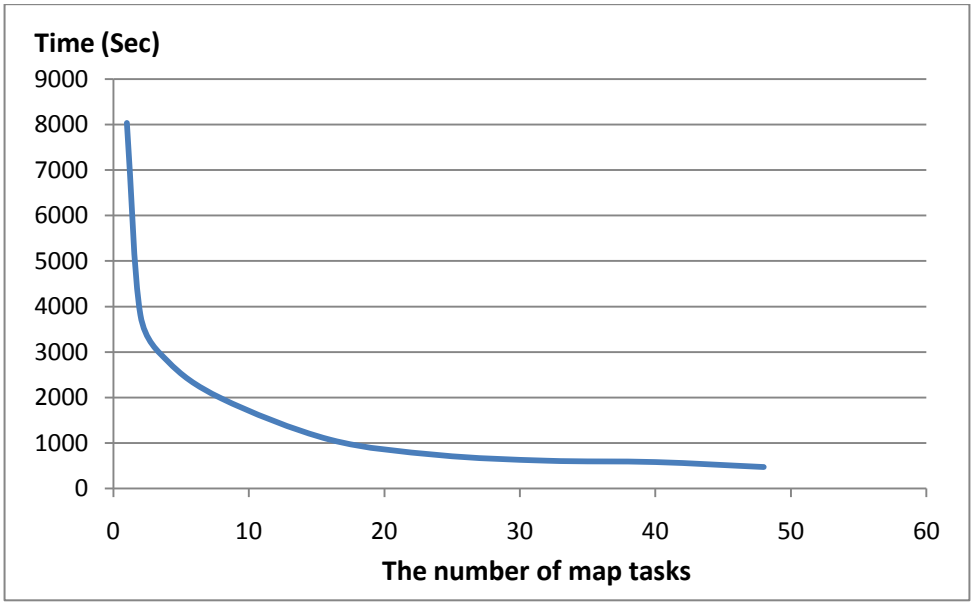


Figure 4.2.1 The performance of the Hadoop cluster. The number of map tasks indicates the amount of computing power dedicated to solve the problem. As the computing resource is increased, the running time of the application is decreased.

A nice improvement on the performance of the application has been observed, when the number of the map tasks is arranged to smaller numbers. The observations show that the running time decreases sharply, when the number of cores is increased. However, any remarkable effect on the performance could not be observed, when the number of the map tasks is arranged to greater numbers. Actually, this is expected because the initialization time will be increased with the number of the map tasks. For example, when the number of the map tasks is set to 48, there will be at least 6 different worker nodes running together on the cluster. They are connected to each other with Ethernet technology. Therefore, initializing and utilizing all of these worker nodes takes significant amount of time comparing to CPU wall time. As a result, the performance of the Hadoop has been dropped.

Hadoop allows the system administrators to set map task number more than the core number the worker nodes have. The performance has been evaluated by setting map task number twice the core number. Furthermore, both numbers are set to the same value and the performance has been evaluated as well. Figure 4.2.2 compares the result of these operations and shows that Hadoop provides better performance under all the experiments with the different core numbers, if the map task number has been set to twice the core number.

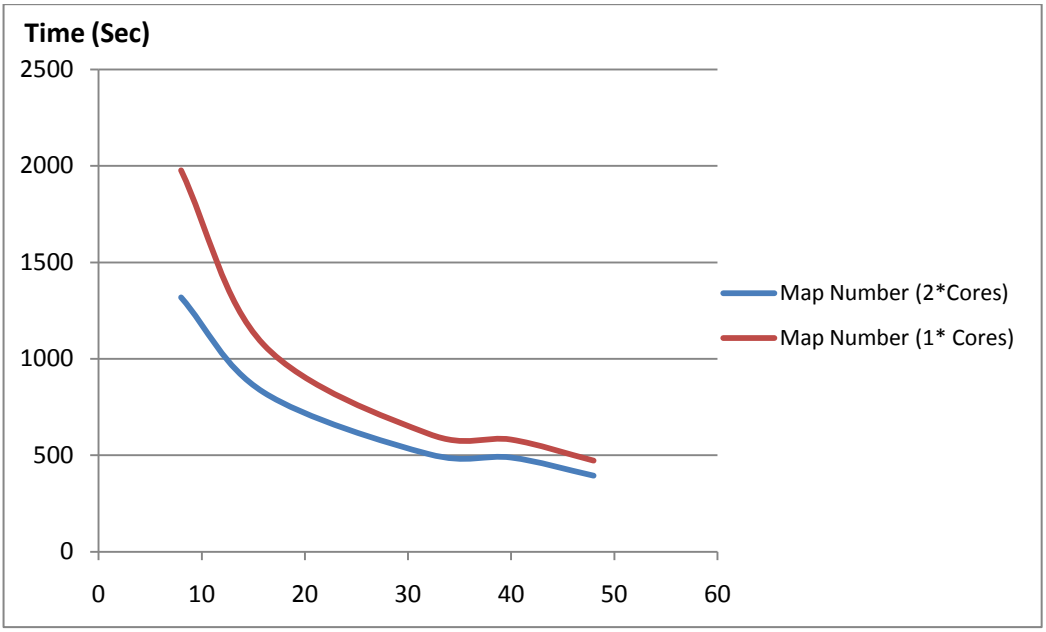


Figure 4.2.2 The performance effect of the number of map tasks on the number of cores on the Hadoop Cluster. Hadoop provides better performance under all the experiments, when the map task number has been set to twice the core number.

It is not a good choice to set the number of map tasks more than twice the core numbers for productions of AMD processors. It causes the worker nodes to be fully loaded and performance lost on the cluster. Table 4.2.1 shows the changes on the performance of the implementation according to the different number of maps, whereas the core number remains constant. There is a performance lost on the cluster, when the number of map tasks is set to three times the number of cores. As a result, it is important to set number of map tasks optimally by taking into account the number of cores.

Table 4.2.1 The relation of map tasks and core numbers on the performance

Number Of Cores	Number of Maps	Time	Iteration Numbers
48	48	7 min 53 sec	17491
48	96	6 min 34 sec	17491
48	144	8 min 2 sec	17491

4.3 The Evaluation of Hadoop Performance by Ganglia

Ganglia is a good monitoring tool to evaluate the performance of Hadoop. In the previous section, the performance of the Hadoop was examined by the running time of the applications over the various numbers of maps and cores. It is mostly based on the information provided by the Hadoop user interface. However, it is not sufficient to understand how well the resources can be utilized. It would neither give any information about the performance of any specific worker nodes in the Hadoop cluster nor about the specific cores in a node. Therefore, it is difficult to detect whether the problem is well partitioned over the clusters and all the cores as well as whether the worker nodes are well utilized by the platform. As a result, Ganglia is used to conduct these detailed analyses.

A number of jobs have been submitted to a cluster formed by setting it to the various numbers of cores and map tasks. Furthermore, all of the running jobs on the cluster have been monitored by Ganglia. Figure 4.3.1 shows one of the results generated by the Ganglia for the jobs running on 64 map tasks over 32 cores. This job has been utilized from four hosts simultaneously (32 cores and 64 GB memory) where the number of map tasks was set to twice the number of cores all the worker nodes have in total.

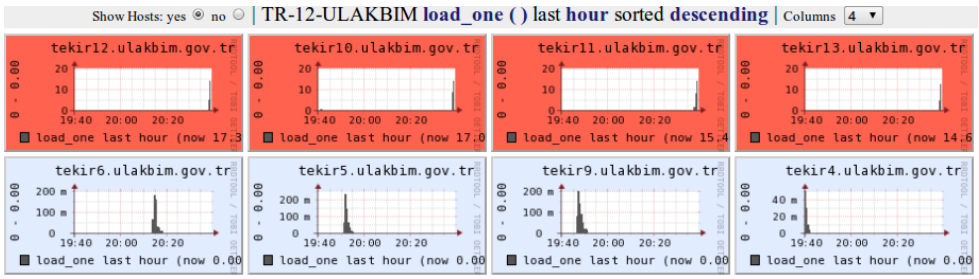


Figure 4.3.1 The load of worker nodes during the running job. It shows that the dedicated computing nodes are utilized fairly and precisely.

The host names of the dedicated worker nodes are tekir10, tekir11, tekir12 and tekir13 where each one runs 16 map tasks in order to perform a large amount of computation in a parallel way. These hosts are marked with a red color which indicates that they are fully loaded. Since the number of map tasks was set to twice the number of cores the worker node has, the loads of these worker nodes were high, as it was expected. Furthermore, Ganglia represents all of the dedicated worker nodes with red colors. This means that Hadoop is capable of utilizing all of the

dedicated resources fairly and precisely. If a part of the dedicated hosts were fully loaded while the other group of hosts was idle, there would be considerable trouble affecting the performance. On the other hand, tekir4, tekir5, tekir6 and tekir9 are the host names represented by the graph. They are in the idle state during the calculation because they are not dedicated to solve this problem by Hadoop. Therefore, they are marked with a blue color which indicates that their loads are very close to zero.

The initialization time is an important issue in the field of high performance computing because it dramatically affects the performance of computing platforms. At the beginning of the application run time, some of the hosts might be idle because these hosts could not be utilized well during the initialization operation. When the initialization operation takes a long time, the running time of the application increases and the dedicated resources cannot be utilized fairly. Ganglia is a very useful tool in order to detect initialization time as well as historical load of the worker nodes.

Figure 4.3.2 presents the CPU load for the dedicated hosts during the running time. The number of map tasks and cores the worker node has were set to the same values. Therefore, the loads of the hosts are not very high. The job was submitted at 17.00 and the graph was obtained as soon as the computation was completed (about 20 minutes). As the figure shows, the Hadoop platform is successful for keeping initialization time short and for utilizing the hosts well because no idle hosts are available during the running time. The other results of the several job submissions with different number of maps and cores show similar results as well.

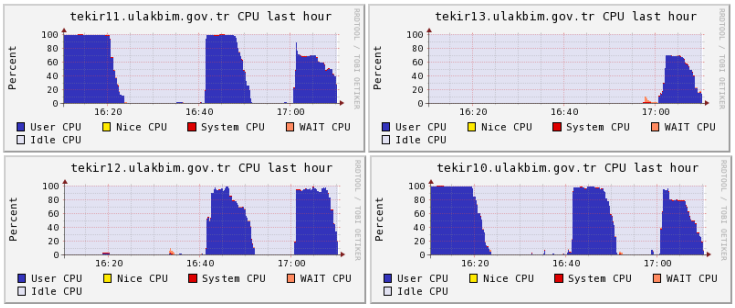


Figure 4.3.2 Historical CPU load of the dedicated hosts. The initialization and completion time are successfully kept short and the hosts are utilized equally.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this chapter, the results of the study were interpreted by presenting their underlying meaning and overall significance. Furthermore, it comprises of the recommendations for future researches in the same area.

5.1 Conclusion

Determining protein functions is one of the most important problems of the post-genomic era. The most classical methods for this task predict function from sequence homology by using programs such as FASTA and BLAST. However, a growing interest in biology has shifted from a study of a single protein or a small complex to an entire proteome generated via the large-scale and high-throughput techniques. Exploiting function information from a whole network of proteins brings about the necessity of high performance computing on a robust computing infrastructure.

Functional Flow is one of the well known methods for predicting protein functions and it requires a large amount of computations according to the size of interaction network and the iteration number for each given protein function. In this thesis, we devised a new algorithm on a parallel and distributed computing platform in order to apply this method to the Hadoop platform. Thus, a new application that enables researchers to study the large proteome generated for the complex organisms has

been implemented in an efficient way. It provides a convenient environment where the researchers can set their scientific parameters properly and can quickly obtain the results of a number of trials quickly.

A protein-protein interaction network can be abstracted as an undirected graph. It is important to know the performance of the Hadoop platform on processing complex graphs because Hadoop is mainly developed for the kind of search engines and text mining applications. In this thesis, we evaluate the application performance by running various numbers of jobs and using a specific tool designed for a distributed computing platform. Utilization of the dedicated computing resources, initialization time, communication overheads and synchronizations were investigated and we present that Hadoop is convenient for bioinformatics problems as well.

For the algorithm implemented in this thesis, partitioning, generating and accumulating flows, and prediction are all written as Map/Reduce jobs. This helps scaling the computations horizontally with the cluster size. We show that our method provides an increasing performance as the number of dedicated resources is increased when the network size is large.

5.2. Future Works

The installed cluster has been formed with a number of computers having 8 cores (AMD CPU's) and the Ethernet technology to enable the computers to communicate with each other. However, the technology on the computing as well as network communication has rapidly evolved. Therefore, the application can be run on a new cluster having the most recent technology. Now, a single computer might have 48 cores and Infiniband technology enables data communication between the computers with 40Gbps speed (QDR). The new generation of the technology might constitute a Hadoop cluster where the application can efficiently be run on a very large proteomics dataset. Especially, the IB technology will dramatically increase the performance of the application because the transfer speed is very important during mapping and reducing operations.

In a single computer, it is not possible to handle all the flows propagated for each of the defined function because of the computer memory restriction. However, a scientist can handle all of these flows and make an operation in a Hadoop computing properly. Thus, they can improve the formula of Functional Flow algorithm. For example, a person can define a new statement in the formula that can control the total amount of flow on the edge. The amount of flows for various known functions which are propagated through the same edge can be arranged according to confidence value of this edge in order to increase the prediction accuracy. Thereby, we might achieve not only a remarkable increase on the computing performance but also more accurate protein function predictions.

The application was one of the embarrassingly parallel implementations in that the problem can be partitioned into a number of small parts and these parts can be processed independently. Hadoop is not only a single computing platform where these kinds of scientific problems can be solved. Grid computing might also be an alternative for the scientists in case they need to run the application on a great number of large and complex datasets which requires thousands of computers. A grid infrastructure is composed of a various number of computing clusters geographically distributed over the world and it can be used for this purpose.

REFERENCES

[1] G. Pandey, M.Steinbach, R. Gupta, T. Garg and V. Kumar. Association Analysis-based Transformations for Protein Interaction Networks: A Function Prediction Case Study. In KDD '07: *Proceedings of the 13th ACM SIGKDD international*, 2007

[2] Schwikowski,B., Uetz,P. and Fields,S. (2000) A network of protein–protein interactions in yeast. *Nat. Biotechnol.*, 18, 1257–1261.

[3] Hishigaki,H., Nakai,K., Ono,T., Tanigami,A. and Takagi,T. (2001) Assessment of prediction accuracy of protein function from protein–protein interaction data. Yeast, 18, 523–531.

[4] Vazquez,A., Flammini,A., Maritan,A. and Vespignani,A. (2003) Global protein function prediction from protein–protein interaction networks. *Nat. Biotechnol*., 21, 697–700.

[5] Karaoz,U., Murali,T.M., Letovsky,S., Zheng,Y., Ding,C., Cantor,C.R. and Kasif,S. (2004) Whole-genome annotation by using evidence integration in functional-linkage networks. *Proc. Natl Acad. Sci*. USA, 101, 2888–2893.

[6] Nabieva E., et al. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics* 2005;21 Suppl. 1:i302-i310.

[7] Lawrence Livermore National Laboratory. Introduction to Parallel Computing. https://computing.llnl.gov/tutorials/parallel_comp/, last visited on 10/11/2009

[8] Anderson NL, Anderson NG (1998). "Proteome and proteomics: new technologies, new concepts, and new words". *Electrophoresis* 19 (11): 1853–61. doi:10.1002/elps.1150191103. PMID 9740045.

[9] Costanzo M, Baryshnikova A, Bellay J, et al. (2010-01-22). "The genetic landscape of a cell". *Science* 327 (5964): 425–431.

[10] Sharan R, Ulitsky I, Shamir R (2007) Network-based prediction of protein function. *Molecular Systems Biology* 3: 1–13

[11] Lord PW, Stevens RD, Brass A, Goble CA (2003) Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. *Bioinformatics* 19: 1275–1283

[12] Vazquez A, Flammini A, Maritan A, Vespignani A (2003) Global protein function prediction from protein–protein interaction networks. *Nat Biotechnol* 21: 697–70

[13] Wagner A (2001) The yeast protein interaction network evolves rapidly and contains few redundant duplicate genes. *Mol Biol Evol* 18:1283–1292

[14] A. S. Foundation. Hadoop core project. http://hadoop.apache.org/, last visited on 20/10/2010

[15] IBM Developer Work. Distributed computing with Linux and Hadoop. http://www.ibm.com/developerworks/linux/library/l-hadoop/, last visited on 22/10/2010

[16] Javier De Las Rivas, Celia Fontanillo. Protein–Protein Interactions Essentials: Key Concepts to Building and Analyzing Interactome Networks. *PLos Computational Biology*. Vol 6:Issue 6 e1000807, 2010.

[17] A Distributed Graph Mining Framework Based On Mapreduce, Master's thesis, Computer Engineering Department, Middle East Technical University, 2010.

[18] Cormen,T.H., Leiserson,C.E. and Rivest,R.L. (1990) Introduction to Algorithms. *MIT Press, Cambridge*, MA.

[19] Mewes,H.W., Frishman,D., Guldener,U., Mannhaupt,G., Mayer,K.,Mokrejs,M., Morgenstern,B., Münsterkötter,M., Rudd,S. and Weil,B. (2002) MIPS: a database for genomes and protein sequences. *Nucleic Acids Res*., 30, 31–34.

[20] Ashburner,M., Ball,C.A., Blake,J.A., Botstein,D., Butler,H.,Cherry,J.M., Davis,A.P., Dolinski,K., Dwight,S.S., Eppig,J.T. et al. (2000) Gene Ontology: tool for the unification of biology. *The Gene Ontology Consortium*. N t. Genet., 25, 25–29.

[21] Running Hadoop On Ubuntu Linux (Multi-Node Cluster). http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/, last visited on 22/10/2010

[22] Yang Q, Lonardi S (2007) A parallel edge-betweenness clustering tool for protein-protein interaction networks.International Journal of Data Mining and Bioinformatics, 1(3):241-247.

[23] Newman, M. and Girvan, M. (2004) Finding and evaluating community structure in networks, Physical Review E, Vol. 69, pp.026113 (15 pages).

[24] David A. Bader, Kamesh Madduri (2008) A graph-theoretic analysis of the human protein-interaction network using multicore parallel algorithms, Parallel Computing 34 627–639

[25] M.L. Massie, B.N.Chun, and D.E.Culler (2004) The ganglia distributed monitoring system: design, implementation, and experience, Parallel Computing, 30, 817-840

[26] L. Salwinski et al.(2004) The database of interacting proteins: 2004 update, Nucleic Acids Research, 32 (Database issue):D449-51