Auto-Scaling of Virtual Resources for Scientific Workflows on Hybrid Clouds

Younsun Ahn
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
ahnysun@sookmyung.ac.kr

Yoonhee Kim*
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
yulan@sookmyung.ac.kr

ABSTRACT

Cloud computing technology enables applications to employ scalable resources dynamically. Scientists can promote large-scale scientific computational experiments over cloud environment. It is essential for many-task-computing (MTC) to certificate stable executions of applications even rapid changes of vital status of physical resources and furnish high performance resources in a long period. Auto-scaling with virtualization provides efficient and integrated cloud resource utilization. Auto-scaling issues have been actively studied as effective resource management in order to utilize large-scale data center in a good shape but most of the auto-scaling methods just easily support performance metrics such as CPU utilization and data transfer latency but seldom consider execution deadline or characteristics of an application. We propose an auto-scaling method that finishes all tasks by user specified deadline. We accomplish our goal by dynamically allocating VMs to maximize resource utilization while meeting a deadline and considering task dependency and data transfer time in workflow application. We have evaluated our auto-scaling method with protein annotation workflow application which tasks are specified as a workflow in hybrid cloud environment. The results of a simulation show the method performs automatically resource allocation actually needed satisfying deadline constraints.

Categories and Subject Descriptors

C.2.4 [Computer Systems Organization]: Distributed Systems

Keywords

Cloud computing, workflow, auto-scaling, hybrid.

1. INTRODUCTION

The appearance of Science Clouds allows scientists to facilitate large-scale scientific computational experiments over cloud environment. Cloud computing enables applications to employ scalable resources dynamically. It is essential for many task computing (MTC) to certificate stable executions of applications even rapid [1] changes of vital status of physical resources and furnish high performance resources in a long period. Thus, it is getting significant to research computational problem solving environment that gives the management of task executions or resources in the large scale of computation. Auto-scaling with virtualization can use efficient integrative utilization of cloud resources for the computational problem solving environment. Technically, auto-scaling can dynamically change the number of Virtual Machine (VM) during execution of an application.

Our previous paper [1] proposed an auto-scaling method to provide efficient resource utilization in a hybrid cloud computing environment. However, proposed auto-scaling algorithm needs to be extended in order to support various patterns of task execution, which are Bag-of-Tasks [2] and workflows. Tasks in Bag-of-Tasks [2] can be scheduled on resources separately from each other while tasks in workflow can be performed in order of dependency pattern. A workflow is commonly represented by a directed acyclic graph (DAG).

This paper proposes an extended version of the auto-scaling method, reflected in patterns of tasks and the requirements of an application based on cloud computing infrastructure. Especially, it automatically allocates resources depending on tasks in workflow on hybrid cloud. We propose an auto-scaling method that finishes all tasks by user-specified deadline. We accomplish our goal by dynamically allocating VMs to maximize resource utilization while meeting a deadline and considering task dependency and data transfer time in workflow application. We have evaluated our auto-scaling method with a specialized computation and data analysis application such as protein annotation workflow application [3] which tasks are specified as a workflow in hybrid cloud environment. The results of a simulation show the method performs automatically resource allocation actually needed satisfying deadline constraints.

The rest of the paper is organized with the sections as follows; we introduce an overview of related works in Section 2. Section 3 explains auto-scaling algorithm, and Section 4 contains contents about a scenario of using auto-scaling algorithm and experiment results are discussed. Finally, we conclude the paper and discuss future work in final section.

2. RELATED WORK

Cloud computing offers endless resources by virtualization technology and facilitates extension and reduction of resources. Auto-scaling issues are currently being discussed and studied as effective resource management. On a cloud service provider side, "Auto-scaling" of AWS [4], Paraleap [5] for Windows Azure [6], and Scalr [7] use a rule-based auto-scaling method which flexibly increases or decreases resources to meet user-defined metrics. Meanwhile, rule-based auto-scaling methods are uncomplicated enough to allocate resources dynamically, it may not be possible

* Corresponding Author

ScienceCloud 2014, June 23, 2014, Vancouver, BC, Canada.
Copyright © 2014 ACM 978-1-4503-2911-8/14/06…$15.00.
http://dx.doi.org/10.1145/2608029.2608036

to satisfy an amount of resources actually needed, then it could possibly cause violation of deadline and cost.

From this reason, on a user side, [8], [9], [10], [11], [12] are the studies of auto-scaling in consideration of deadline of applications or cost for resource usage. [8] proposes an auto-scaling method minimizing resource usage cost for Bag-of-Tasks [2] jobs. It is connected with horizontal scaling which adds or removes the number of VMs and vertical scaling which expands or reduces the size of a VM. However, it is still deficient for accomplished resource requirements of dynamic workloads because it is lack of the consideration of resource usage during execution of an application. There are few studies considering workflow scheduling. [9] proposes an algorithm to minimize execution costs while meeting deadline for a workflow. Reference [10] respects to finish the execution of jobs within deadlines at minimum financial cost using their auto scaling method. However, they use only public cloud, which means that they just consider simple environment. [11], [12] facilitate the execution of workflow applications using their auto-scaling methods on Grids, which may be changed to adapt to the Cloud computing environment. [11] proposes an algorithm that minimizes execution cost, that is processing cost and data transmission cost within deadline. [11] divides the workflow into partitions and assigned each partition a sub-deadline, thus [11] can minimize execution time for the entire workflow. [12] proposes an efficient scheduling algorithm for dependent jobs. Also, [12] considers communication cost about data transfer time. We propose an algorithm for workflow referring to Reference [11]'s and [12]'s workflow scheduling algorithms.

In this paper, we propose an extended version of an auto-scaling method based on our previous research [1] which enables dynamic resource allocation considering the types of jobs from Bag-of-Tasks [2] as well as the workflow and the characteristics of an application. Auto-scaling method can automatically allocate cloud resources considering task dependency and data transfer time in workflow application.

3. AUTO-SCALING ALGORITHM

We extend [1]'s auto-scaling algorithm which can perform only tasks in Bag-of-Tasks. Our auto-scaling algorithm can schedule tasks in workflow. Auto-scaling technique can discover delay and deadline violation to comparing actual start time and estimated start time of running tasks. Moreover, monitoring perform every regular term. Algorithm's assumption and notation are referred to [1]. Some notations for the workflow scheduling algorithms are defined at the below:

- EFT_{VM}: Estimated finish time of a VM.
- EST_{VM} : Earliest start time of a VM.
- ET_{VM}: Execution time of a task on a VM.

Algorithm 1, Run-time Scaling is extended based on [1]. We propose algorithm 2, Workflow Scheduling algorithm to deal with tasks in workflow. In the reference [1], Run-time Scaling Algorithm has two polices and depicts our general auto-scaling method. Additionally, we extend three policies by adding workflow scheduling. Tasks are scheduled with applying one of the three policies such as Performance-oriented Scheduling (line 5), cost-oriented Scheduling (line 8) and Workflow Scheduling (line 11) of SLA (Service Level Agreements). Tasks in Bag-of-Tasks [2] are sorted as descending order based on their execution time, while tasks in workflow are sorted as sequential order.

Algorithm 1 – Run-time Scaling

```
Input – An application,
        SLA={a policy P, a deadline D [, minimum performance requirement
        minPM ]}
Output – Scaling decision S = { toStartUp, toShutDown }
         Scheduling decision S = { tasks → VMs}
 1: SCALING ← TRUE;
 2: while (true)
 3:     if SCALING is TRUE
 4:         switch P
 5:         case Performance:
 6:             Sort waiting tasks in decreasing order of execution length;
 7:                 S ← PerformanceOrientedScheduling(
                                sortedTasks, D, minPM);
 8:         case Cost:
 9:             Sort waiting tasks in decreasing order of execution length;
10:                 S ← CostOrientedScheduling(sortedTasks, D);
11:         case Workflow:
              Sort waiting tasks in sequential order;
12:                 S ← WorkflowScheduling(sortedTasks, D);
13:         each vm where status is running
14:         if no running/waiting tasks on vm then
15:           destroy the vm
16:         send scaling decisions to DRMS
17:         send scheduling decisions to JES
18:         waitForNextInterval();
19:         SCALING ← SLAMonitoring(runningTasks, D);
```

Algorithm 2 defines our Workflow Scheduling algorithm. In this algorithm, DTT (Data Transfer Time) means data transfer time. Proposed Workflow Scheduling algorithm is based on a PCH algorithm [12]. When our scaling method tries to allocate cloud resources, it followed the performance-oriented policy. We could get tasks on a critical path and group of tasks using PCH algorithm [12] and then each group can be scheduled. The total execution time of critical path in a private cloud resource is calculated and set to a deadline value, also additional time is add to the deadline value.

First, tasks on a critical path are scheduled on a same resource, which can execute all tasks in a critical path (line 2). Every time we try to schedule tasks, it is a rule to choose private cloud resource prior to public cloud. It is important to consider task dependencies and data transfer time in workflow. Each task could get an EFT (Estimated Finish Time) of a parent task and set an EST (Earliest Start Time) value to EFT of a parent task in order to consider order of tasks (line 4-5). With an executing on a same resource of tasks on the critical path, a communication overhead is reduced. Parent task and child task are performed on a same VM, data transfer time is zero.

When tasks which are not on a critical path are scheduled to VMs, tasks check a parent task's state. If parent tasks are allocated to cloud resources, child tasks could be allocated (line 6). Unless parent tasks are scheduled cloud resources, child tasks must wait to be allocated. As mentioned above, each task could get an EFT of a parent task and calculate an EST value considering EFT of a parent task and data transfer time. In case there are a number of parent tasks, a task chooses latest EFT among them and calculates an EST value with EFT of a parent task and data transfer time. (line 7-10 (private), line 12-16 (public)).

Algorithm 2– workflow scheduling algorithm
Input –Waiting jobs of the application,
Output –Scheduling decision $S = \{ tasks \rightarrow VMs\}$

```
 1:  for each task do
 2:  if task in a critical path then
 3:      if There is no VM running then
 4:          VM ← find a vm on which all tasks in a critical path can
                 run within the D;
 5:      else
 6:          VM ← current VM for a critical path
 7:      end if
 8:          $EST_{VM}$ ← related previous job $EFT_{VM}$ and previous task's DTT;
 9:          $EFT_{VM} \leftarrow EST_{VM} + ET_{VM}$;
10:  else
11:      VM ← find a vm on which task can start the
                 fastest within the D; // find a private vm
12:      $EST_{VM}$ ← related previous task $EFT_{VM}$
                 and previous task's DTT;
13:      $EFT_{VM} \leftarrow EST_{VM} + ET_{VM}$;
14:      if VM is null //no private VM available
15:          VM ← find a cheapest vm on which task can run within the D;
                 // find a public vm
16:          $EST_{VM}$ ← related previous job $EFT_{VM}$
                 and previous task's DTT;
17:          $EFT_{VM} \leftarrow EST_{VM} + ET_{VM}$;
18:      end if
19:      schedule task to VM;
20:      continue with the next task;
21:  end if
```

4. SCENARIO AND EXPERIMENTS

We use CloudSim [13] to simulate protein annotation workflow in workflow. We simulate our proposed auto-scaling algorithm in hybrid cloud environment that is merged private cloud and public cloud resources. In this experiment, we use four private clouds have 330 MIPS and public clouds (Amazon EC2) have 600 MIPS or 2400 MIPS.

Protein annotation workflow [3] application has dependent tasks and needs to deal with large information. Protein annotation workflow [3] developed by London e-Science Center. Protein annotation workflow [3] application has fifteen services. Protein annotation workflow [3] application consists of several steps to fulfilling service. Several services are sequentially performed. Every service in a workflow generates output data required by its child services as inputs in Figure 1.

Among services, we specially consider HMMer, IMPALA, BLAST, PSI-BLAST and PSI-PRED services. Each service's explanation refers to [14], [15], [16], [17], [18]. In our experiment, we concentrate input data and output data in order to consider data transfer time. HMMer, IMPALA, BLAST, PSI-BLAST and PSI-PRED services are required large input data. Accordingly, in case of considering data transfer time, all values of data transfer time are not equal. Values which are required large input data are given a weighting. PSI-BLAST service is required not only large input data, but also large output data, because next PSI-PRED service needs to all output data of the PSI-BLAST. Therefore, values which are required large input data and output data are given a weighting.

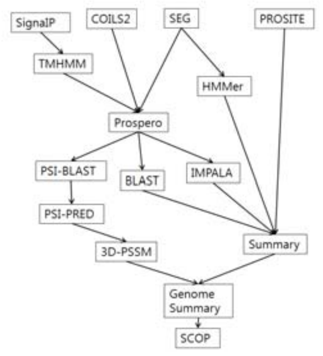

Figure 1. e- protein service structure [3].

Figure 2 shows a simple form of protein annotation workflow [3] in Figure1 representing a number of tasks. The number of tasks is fifteen and we use a length of tasks that are featured on parentheses in Figure 2. Each task has task dependencies. A value on a bridge between tasks represents a data transfer time. Parent task and child task are performed on a same VM, data transfer time is zero. While, they are performed on a different VM, data transfer time is considered with the value. The I/O data of the workflows range from 1.2GB to 29.2GB. The available network bandwidth between services is 100Mbps.

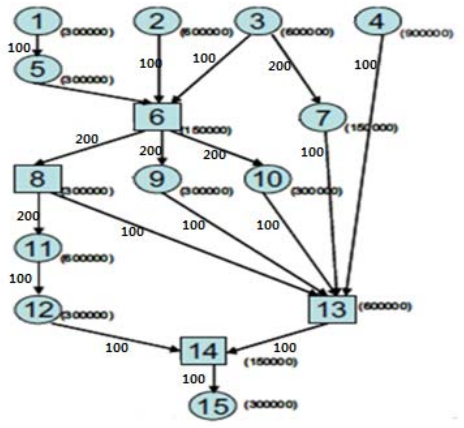

Figure 2. Protein annotation workflow [3] structure.

We compare experiments with different data transfer time. First case, a fundamental data transfer time is 100 seconds and some data transfer time have weighting value that is a 200 seconds. Second, an essential data transfer time is 800 seconds and some data transfer time have weighting value that is a 900 seconds. Third case, a radical data transfer time is 2000 seconds and some data transfer time have weighting value that is a 2400 seconds. Figure 3 shows a result of the number of VM with protein annotation workflow structure (Figure 2) based on the workflow scheduling algorithm.

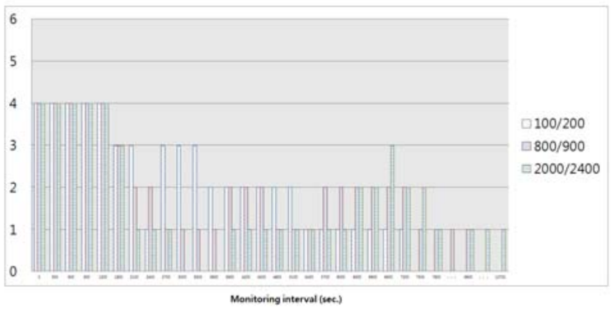

Figure 3. The result of auto-scaling representing the number of VM

Figure 3 represents the performance of the proposed auto-scaling method comparing with different data transfer time. The proposed auto-scaling algorithm successfully performs automatically allocating tasks in workflow considering task dependency and data transfer time. In Figure 3, a dot pattern graph represents data transfer time is 100 seconds or 200 seconds. Entire tasks are completed within 7300 seconds. While, a pattern of diagonal graph shows all tasks are finished within 9600 seconds. It is the result of 800 seconds or 900 seconds data transfer time. Last horizontal pattern graph shows the result of 2000 seconds or 2400 seconds data transfer time case. The dot pattern graph shows that it uses more VMs than a pattern of diagonal graph does in the middle of execution time because tasks in the dot pattern graph can reduce waiting time on account of short data transfer time. The dot pattern graph uses relatively little VMs in the latter half of execution time. In a pattern of diagonal graph, tasks should wait to be allocated until long data transfer time is finished, so they take longer execution time and need little VMs than the dot pattern graph. Horizontal pattern graph shows as similar as other cases at the beginning of the execution time. It takes longest execution time among all cases. Also, it is efficient to allocate same VM for tasks which have task dependency and have long data transfer time. In case data transfer time is short, the auto-scaling method can maximize resource utilization in order to allocate tasks to VMs as soon as possible.

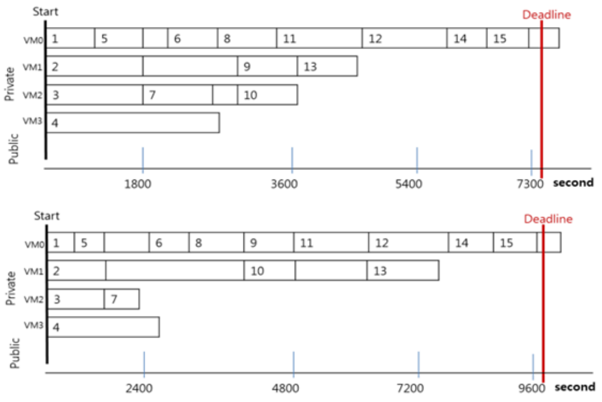

Figure 4 (a). Scenario of data transfer time 100/ 200 seconds.

Figure 4 (b). Scenario of data transfer time 800/ 900 seconds.

Figure 4 represents scenarios using auto-scaling method with Protein annotation workflow [3] application. In Figure 4 (a), we assumed that data transfer time is 100 seconds or 200 seconds. On the other hand, data transfer time is 800 seconds or 900 seconds in Figure 4 (b). Tasks are scheduled in sequential order. Private cloud resources can be considered for tasks in the first priority. If tasks cannot be finished in private cloud resources within deadline, we could arrange public cloud resources for tasks. In Figure 4 (a), to care task dependency, tasks in a critical path choose a VM that can run all tasks in a critical path. Task #1, #5, #6, #8, #11, #12, #14, and #15 are allocated on same VM that is VM #0. Task #2, #3, #4 are not included in a critical path and don't have previous task which related dependency, so they can be allocated in parallel on new VMs. Task #6 confirms EFT of task #5, #3, and #2 and then we add data transfer time and latest EFT among them. As a result, we can get EST of task # 6.Task #7 can be scheduled on VM1, but it is allocated on VM #2 which performs dependency task, that is task #3, in order to reduce a data transfer time. A VM #2 can allocate tasks for 1800 to 2250 second. However, task #8, #9 and #10 cannot be allocated on VM until task #6 is finished so, VM #2 suspends during that time. This result can represent scale-in. Task #4 is finished on a VM #3 and then other tasks do not run on the VM #3. Thus, scale-in occurs. Task #8, #9 and #10 have task dependency on task #6, so they consider data transfer time between task #6. Task #8 can be scheduled on same VM which is allocated to task #6 and data transfer time can be determined 0 second, thus as soon as task #6 is finished, task #8 is allocated on VM. After task #6 and data transfer time is finished, Task #9 and #10 can be allocated. They can be allocated in parallel on VMs. Task #13 has five parent tasks which are task #4, #7, #8, #9, and #10. Task #10 is completed in latest of them all. Therefore, task #13 can be allocated when a task #10 and data transfer time between task #10 is finished. Task #14 is waited until task #13 and task #12 is finished and data transfer time is finished. Finally, task #15 is finished within a deadline. All tasks are finished and meet the deadline.

We supposed that data transfer time is 800 seconds or 900 seconds (as shown in Figure 4 (b)). From tasks #1 to #8 are allocated to same VMs like Figure 4 (a). However, Task #9 and Task #10 are not allocated to resources which are allocated in Figure 4 (a). After task #6 is completed, task #8 is scheduled on VM #0. Task #9 and #10 are waited to fulfill data transfer time and then they can be allocated at 3950 seconds. At that time, All VMs can allocate tasks, so task #9 is allocated on VM #0 and task # 10 is scheduled on VM # 1.

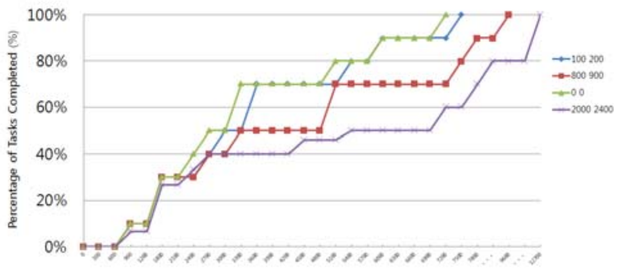

Figure 5. The percentage of tasks completed.

Figure 5 shows the percentage of tasks completed for the Protein annotation workflow [3] application at every interval. Figure 5 represents comparing the percentage of tasks completed by changing data transfer time. If data transfer time is short, the auto-scaling method can complete tasks as fast as possible by automatically allocating VMs. In Figure 5, a triangle line is a result that is not considering data transfer time. Diamond line represents that data transfer time is 100 seconds or 200 seconds. Dot line is a result which data transfer time is 800 seconds or 900 seconds. X line takes 2000 seconds or 2400 seconds in order to transfer data.

Triangle line shows tasks are finished fastest within 7200 seconds among lines. Data transfer time does not influence the execution time up to 2100 seconds. Until 2100 seconds, three lines look like similar style. The dot line is performed faster than any other lines at the end of the execution time. X line shows that tasks are performed 50% tasks during half of the execution time. X line takes long time to transfer data. It is observed that increasing the data transfer time dose affect execution time and resource utilization significantly.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed auto-scaling method that effectively managed applications in workflow type in hybrid cloud computing. We conducted experiments with a scientific application such as Protein annotation workflow [3] application.

In the simulation with protein annotation workflow [3], scale-in and scale-out that is auto-scaling were automatically performed within its deadline. The method expands or contracts resources dynamically by considering task dependency and data transfer time. Resources can be used efficiently as needed using proposed auto-scaling method.

Furthermore, we plan to add diverse policies such as semantic policy considering characteristic of application.

A.1 Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A3007866)

6. REFERENCES

[1] Hyejeong Kang, Jung-in Koh, Yoonhee Kim. 2013. *A SLA driven VM Auto-Scaling Method in Hybrid Cloud environment.* APNOMS, Hiroshima, Japan, 25-28.

[2] W. Cirne, F. Brasileiro, J. Sauve, Na. Andrade, D. Paranhos, E. Santos-Neto, R. Medeiros. 2003. *Grid Computing for Bag of Jobs Applications*. Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, September 2003.

[3] A. O'Brien, S. Newhouse and J. Darlington. 2004. *Mapping of Scientific Workflow within the e-Protein project to Distributed Resources*. In UK e-Science All Hands Meeting, Nottingham, UK. September. 2004.

[4] Amazon Web Service, http://aws.amazon.com/

[5] Windows Azure, http://www.windowsazure.com/

[6] Paraleap, https://www.paraleap.com/

[7] Scalr, http://scalr.com/

[8] S. Dutta, S. Gera, A. Vermam, and B. Viswanathan. 2012. *Smartscale: Automatic application scaling in enterprise clouds*. 5th IEEE International Conference on Cloud Computing (CLOUD)(June, 2012), 221-228

[9] L. Bittencourt, and E. Maderia. 2011. *HCOC: A Cost Optimization Algorithm For Workflow Scheduling in Hybrid clouds. J. Internet Services and Applications*. 2 (December. 2011), Springer-Verlag, 207-227.

[10] M. Mao, and M. Humphrey. 2011. *Auto-scaling to minimize cost and meet application deadlines in cloud workflows.* High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. November 12-18, 2011.

[11] J. Yu, R. Buyya, and C. K. Tham. 2005. *Cost-based scheduling of scientific workflow applications on utility grids*. e-Science and Grid Computing, 2005 First International Conference on IEEE, 8-147.

[12] L. F. Bittencourt, and E. R. Madeira. 2008. *A performance oriented adaptive scheduler for dependent tasks on grids.* Concurrency and Computation: Practice and Experience. 20, 9 (2008), 1029-1049.

[13] Rodrigo N., Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. 2011. *Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*. Software: Practice and Experience, 41, 1 (2011), 23-50.

[14] HMMER, http://hmmer.janelia.org/

[15] IMPALA, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html

[16] Johnson, M. Zaretskaya, I. Raytselis, Y. Merezhuk, Y. McGinnis S. Madden T. L. 2008. *NCBI BLAST: a better web interface*. Nucleic Acids Research 36 (July. 2008), W5-W9.

[17] Bergman, Nicholas H, Medha Bhagwat, and L. Aravind. 2007. *PSI-BLAST Tutorial.*

[18] PSI-PRED, http://bioinf.cs.ucl.ac.uk/index.php?id=779