PROCEEDINGS OF SCIENCE

PoS(ISGC 2013)037

Harness public computing resources for protein structure prediction computing

Wenjing Wu[1]
Computing Center, Institute of High Energy Physics, CAS
19B Yuquan Road, Beijing, China
E-mail: wuwj@ihep.ac.cn

Gang Chen
Computing Center, Institute of High Energy Physics, CAS
19B Yuquan Road, Beijing, China
E-mail: cheng@ihep.ac.cn

Wenxiao Kan
Computing Center, Institute of High Energy Physics, CAS
19B Yuquan Road, Beijing, China
E-mail: kanwx@ihep.ac.cn

David Anderson
Space Sciences Laboratory, University of California, Berkeley
E-mail: davea@ssl.berkeley.edu

Francois Grey
CNMM, Tsinghua University
E-mail: francois.grey@cern.ch

1

Jin Li
Institute of Computing Technology, CAS
E-mail: lijin01@ict.ac.cn

Dongbo Bu
Institute of Computing Technology, CAS
E-mail: dbu@ict.ac.cn

Protein structure prediction is a CPU intensive activity. Although algorithms have improved over time, the amount of CPU required for predicting all the existing protein sequences is still beyond the capacity of any single computer center. At the same time, volunteer computing has been evolving into a widely used technology for scientific computing. This paradigm has enabled previously infeasible science research. In this paper, based on our practical experiences, we will present how we ported the protein structure prediction software TreeThreader to a volunteer computing platform, and achieved the high throughput by designing its computing model based on the features of volunteer computing and the software itself. Running TreeThreader on this platform has achieved 1.27TFLOPS sustained processing rate on about 2000 active volunteer hosts.

The International Symposium on Grids and Clouds (ISGC) 2013
March 17-22, 2013
Academia Sinica, Taipei, Taiwan

PoS(ISGC 2013)037

1. Introduction

1 . 1 Public Resource Computing

The world's computing power and disk space is no longer primarily concentrated in supercomputer centers and machine rooms. Instead it is distributed in hundreds of millions of personal computers and game consoles belonging to the general public. Volunteer computing uses these resources to do scientific computing. This paradigm enables previously infeasible research. The potential resource of this paradigm is very large. The number of Internet-connected PCs is growing rapidly, and is projected to reach 1 billion by 2015 [1]. Together, these PCs could provide many ExaFLOPs of computing power.

1.2 Protein Structure Prediction and TreeThreader

The understanding of protein structures [2] [3] is essential for a complete understanding of life processes at the molecular level. Currently, over seven million protein sequences are deposited in the UniProtKB/TrEMBL database, but only 50,000 of them have experimentally solved structures.

Threading is the leading methods for protein structure prediction, and it is exceedingly time-consuming because the query sequence needs to be aligned to all templates in the database. TreeThreader is a new practical threading program, which can take pairwise interaction into consideration. It has been proved that the general case of the problem (in which all pairwise contacts are considered) is NP-hard. So, TreeThreader employs nested graphs to describe contacts with templates (like covariance model for RNA secondary structure analysis). It outperforms other methods such as HHpred.

With "TreeThreader", it takes about 2 seconds to align an average protein sequence with one template on an average single 2 GFLOPS CPU, so it needs about 27.7 hours to align it with all 50,000 templates and get the predicted structure. Hence it takes about 22,196 years on a single CPU core to finish the prediction of all currently unknown protein structures. So even with TreeThreader, the quantity of CPU power required to compute the large amount of protein sequences is beyond the capacity of any computer center.

The rest of the paper is organized as follows. Section 2 gives an introduction to the BOINC platform. Section 3 presents the computing model design of TreeThreader on BOINC platform. Section 4 gives details of the implementation including the control program, results validation, locality scheduling, and job submission system. Section 5 presents the results and Section 6 the conclusions.

PoS(ISGC 2013)037

2. BOINC

BOINC [4] (Berkeley Open Infrastructure for Network Computing) is a platform for volunteer computing. BOINC is being used for applications in physics, molecular biology, medicine, chemistry, astronomy, climate dynamics, mathematics, and the study of games. There are currently about 50 BOINC based projects and about 350,000 volunteer computers performing an average of over 9.1 PetaFLOPS [5].

2.1 BOINC Features

BOINC harnesses heterogeneous and untrusted computing resources, so it has unique features compared to other types of distributed computing.

- BOINC provides high level transparency over heterogeneous platforms of different hardware and operating systems.
- BOINC deals with distrusted personal computers using redundant computing and validation. Validation is application specific, and only consistent results are considered to be valid.

Running an application on volunteer computing resources also imposes several limitations:

- BOINC assumes no runtime environment on volunteer computers, so applications should not rely on any dynamical libraries; therefore the application needs to be statically compiled.
- The application runs at a low priority on volunteer computers, so it is only suitable for computation with loose deadlines and application needs to support checkpointing of long running tasks.
- Public computers have limited Internet bandwidth, so the computation should not require a large amount of input/output data.
- BOINC clients do not supports communicate between each other, so it is not suitable for distribution application which needs internal communication.

TreeThreader software is written in C, can be compiled to be runtime environment independent with some effort. The supports checkpoint. The computation is CPU intensive. The comparisons of a sequence to each template are independent , hence a computing task can be split into sub tasks and run on multiple hosts in parallel. So the TreeThreader software is a suitable application to run on volunteer computing resources.

2.2 BOINC Architecture

As shown in Figure 1, the BOINC middleware consists of the server and client components, and the communication between the two sides is via XML over HTTP . The communication is always initiated by the client side, ie. , requesting jobs from the server side, downloading input files and uploading output files. There are several daemons running on the server side, including the feeder, scheduler, transitioner, validater and assimilater.

The scheduler handles requests from BOINC clients. Each request includes a description of the host, a list of completed instances, and a request for additional work, expressed in terms of the time the work should take to complete. The reply includes a list of jobs.

The feeder streamlines the scheduler's database access. It maintains a shared-memory segment containing 1) static database tables such as applications, platforms, and application versions, and 2) a fixed-size cache of unsent jobs. The scheduler finds jbos that can be sent to a particular client by scanning this memory segment.

The validater examines sets of results and selects canonical results. It includes an application-specific result-comparison function.

The assimilater handles newly-found canonical results. It includes an application-specific function which typically parses the result and inserts it into a science database.

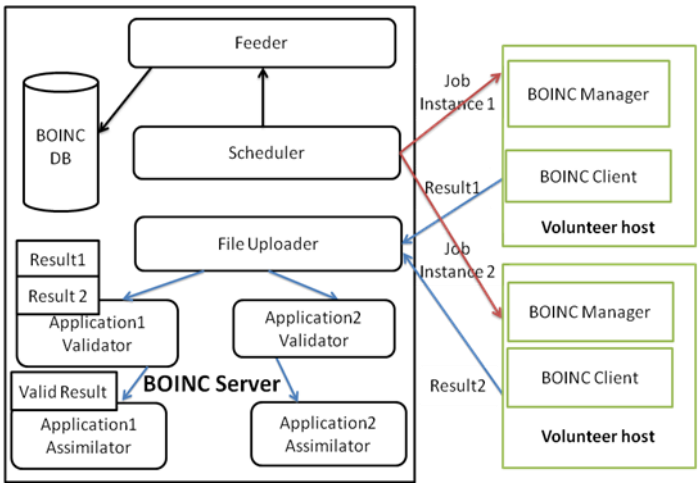

Figure 1: BOINC Software Architecture

3. Computing Model Design

The total input files (all existing protein templates) for a protein structure prediction computing task is about 5GB with 50,000 templates, and it takes about 28 hours to finish on a single CPU core. As discussed in section 2.2, computing on volunteer resources requires a high ratio between CPU time and data transfer time and also a small amount of input/output data, so placing 5GB input data on a single host is impractical for BOINC. Fortunately, in protein structure prediction, the comparisons of a sequence against the templates are independent from each other, so the computing task can be split into small sub-tasks based on subsets of templates, and the results of each sub-task can be merged together to construct the structure of the protein.

Here 50,000 templates are split into 32 sub packages, with each package having about 1,600 templates and a size of 50MB after compression, so on an average volunteer host, it takes about 30 seconds to download one template package, and about 20 minutes of CPU time to process the comparisons. The corresponding results are omuch smaller , usually around 2MB for a package after compression, so the time taken to upload the results can be ignored.

In this way, each computing task (predicting one protein sequence) is split into 32 sub-tasks with each sub,task processing one template package. Each sub,task is a job on

PoS(ISGC 2013)037

BOINC server, and the BOINC scheduler can distribute the 32 sub-tasks to multiple hosts which request jobs. The results from each sub-task will be merged together by the Job Submission Interface when all the sub-tasks are finished. By splitting the computing task and distributing the sub-tasks on multiple hosts, the execution time of one computing task is significantly reduced.

4. Implementation

In order to port TreeThreader on BOINC, a few customizations needed to be done on the BOINC server side.

4.1 Control Program

On the BOINC client, the actual application such as TreeThreader needs to be able to communicate with the core client so the BOINC client can suspend/resume/abort the process and check the status of the process. BOINC provides a program called "Wrapper" which can run the other applications as child processes, and handles the communication between the applications and the BOINC client.

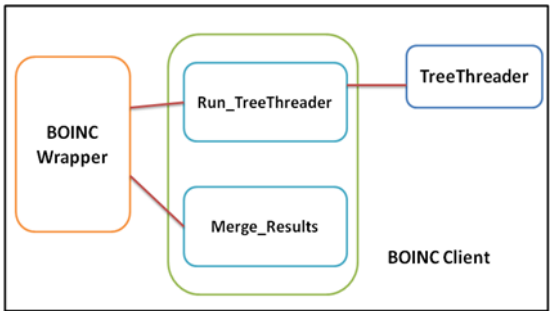

Figure 2: TreeThreader Control Program tree

As shown in Figure 2, a sequence prediction sub-task includes two functions: running the TreeThreader software and merging the results. Two programs are used for this purpose They are run as child processes and controlled by the BOINC Wrapper,. "Run_TreeThreader" sets up the necessary running environment, edits the configuration file, uncompresses the templates, and start the "TreeThreader" application with its input files. Upon the finish of "Run_TreeThreader", "Merge_Results" scans the result files corresponding to the templates and compress the results which will later be uploaded by BOINC client to the server.

4.2 Results Validation

Several factors [6] can lead to inconsistent results on volunteer computing resources: the heterogeneity of hardware and operating systems can produce discrepant results in float point calculation, and malicious owners of the hosts can falsify the results. To address this problem, an application specific validater needs to be implemented to determine whether

PoS(ISGC 2013)037

results from replicated jobs agree with each other, and one of the consistent results will be marked as "canonical result" for this job.

In the case of TreeThreader, it generates one result file corresponding to each template it uses to compare with the sequence, so each sub task produces the same quantity of result files as the quantity of templates it processes, and all the result files are compressed into one file and uploaded to the BOINC server.

The result files are in plain text format, and do not have discrepancy over heterogeneous platforms, but uncompressing all the result files for every single sequence and doing a bitwise comparison is very time consuming, so a very simple mechanism is used to validate the results: comparing the size of the compressed result files of a sequence. This is a very loose validation; however, post processing of the full results on another server will pick out the invalid results.

4.3 Locality Scheduling

When a host makes a request of jobs, BOINC scheduler [7] checks the shared memory which is managed by the BOINC feeder to decide what jobs match the request. BOINC scheduler takes general information such as estimated FLOPS, disk size, and memory size of a job into account to match the job request.

In TreeThreader, each sub-package of templates corresponds to a sub-task of a sequence prediction and can be reused by another sub-task of another sequence. To save the bandwidth, template package files are marked as "sticky" on BOINC client, so they will not be removed by the BOINC client after being used by one sub-task so another sub-task can reuse it later. However, this still means if a host gets random sub-tasks from the scheduler, it will tend to "cache" all the template packages on the client disk eventually. Therefore, the client will use a large amount of storage and initial bandwidth to download the template packages. To solve this problem, a "locality scheduling" mechanism is implemented in the BOINC scheduler.

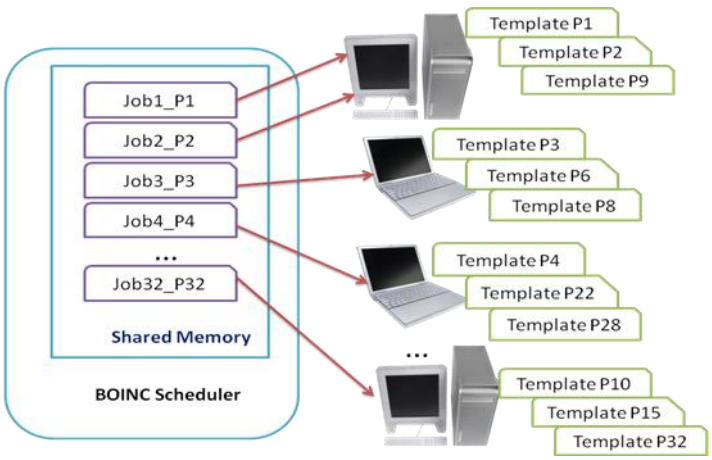

Figure 3: TreeThreader Locality Scheduling

As shown in Figure 3, when a BOINC client sends a job request to the scheduler, it includes a list of the sticky files it stores The BOINC scheduler can then scan the available jobs in the share memory, and select the jobs which use these sticky files as input data, and

PoS(ISGC 2013)037

combine this with other matching information to decide what jobs could be sent to the client. For example, if client A indicates in the job request that it has template package file P1, P2 and P9, then the BOINC scheduler will preferentially send jobs which require file P1, P2 and P9 to this client. The initial number of template package files is determined by the number of parallel jobs allowed to be run on the client.

4.4 Job Submission

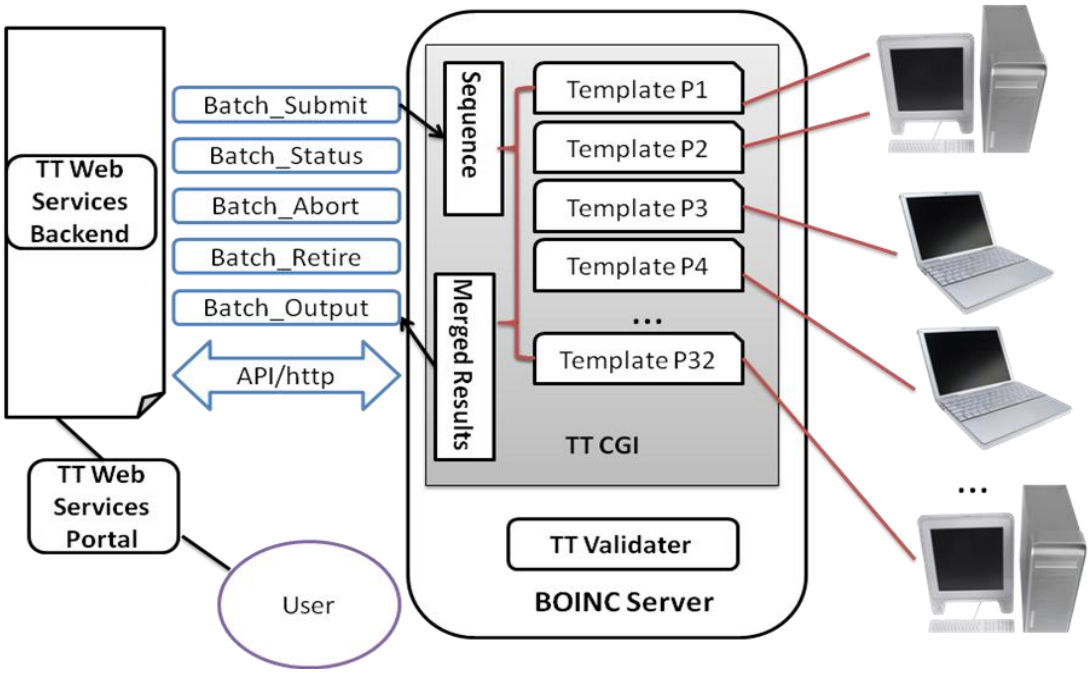

Figure 4: TreeThreader Job Submission Diagram

The job submission system consists of 4 components as shown in Figure 4:

1) The Web Portal [8] from which scientists can submit protein sequences, check the process status and displays the structure of finished proteins [9].
2) The backend of the Web Portal which verifies/preprocesses the sequence and submits valid sequences to the BOINC server.
3) A set of remote job submission interfaces which enables users/applications to interact with BOINC server through HTTP . Five interfaces are provided: Batch_Submit which allows the client to upload a sequence to the BOINC server to generate a batch of jobs and returns the batch ID for this sequence; Batch_Status which queries the status of a submitted batch; Batch_Abort which allows to abort a submitted batch; Batch_Retire which allows to retire a completed batch and remove all its associated files; Batch_Output which downloads the merged results of a finished batch.
4) A CGI program runs on BOINC server which responds to HTTP requests of remote job submission. Upon receiving a Batch_Submit request, the CGI program takes the sequence and creates a batch of 32 jobs based on the 32 template packages on BOINC

PoS(ISGC 2013)037

server. Upon receiving a Batch_Output request, the CGI program picks out all canonical results (results being validated) of the batch and compresses the results into one file for the client to download.

5. Results

5.1 Process Rate

The TreeThreader application was ported and officially launched on the BOINC based volunteer project CAS@home [10] in November 2012. As of August 2013, it has gained 3.5 Million effective CPU hours from the volunteer computing resources including home/office PCs and laptops as shown in Figure 5, and it has finished predicting 27 thousand protein structures since its launch as shown in Figure 6 (a batch of jobs is equivalent to a protein sequence prediction).

Both Table 1 and Table 2 show the current available and achieved resources from CAS@home project [11].

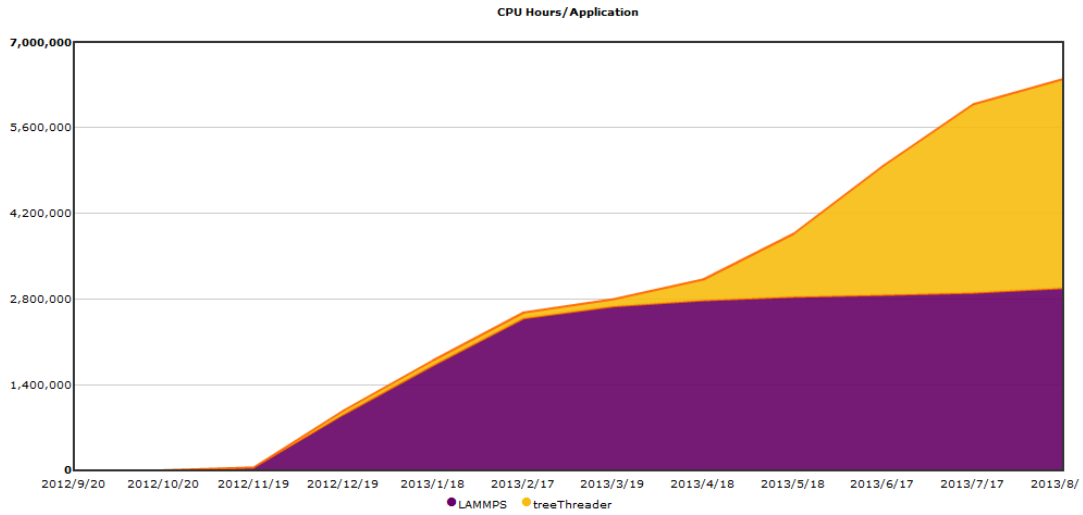

Figure 5: Effective CPU hours on CAS@home

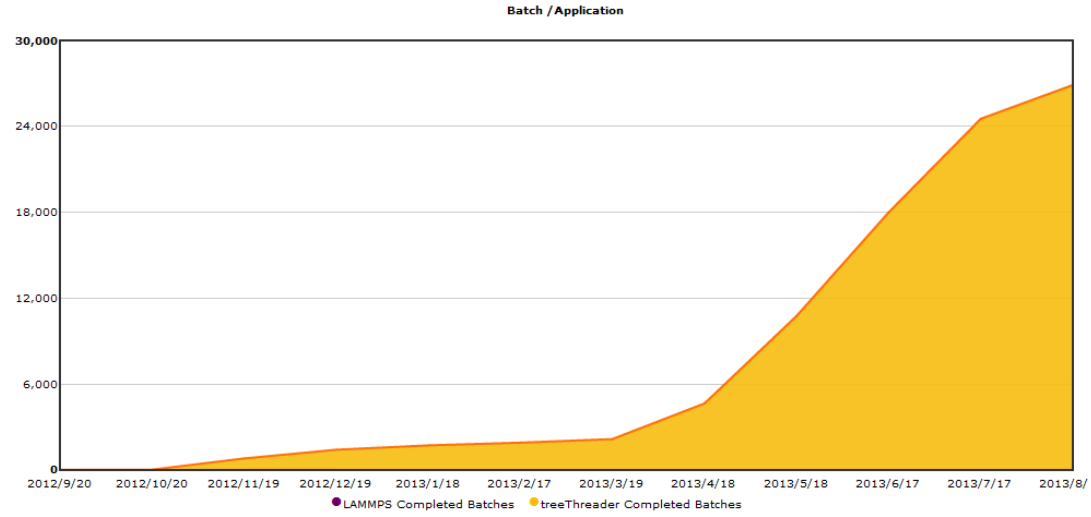

Figure 6: Finished Protein sequences on CAS@home

	Registered	Active
User	16,454	1,183
Host	21,941	1,752
CPU Cores	71,351	8,416
Available FLOPS	174.7TeraFLOPS	22.78TeraFLOPS

Table 1: Resource Statistics of CAS@home

RAC(Average Credit/Day)	28,746
Real Time FLOPS	1.27TeraFLOPS
Project Total CPU Hours	13,698,551

Table 2: Project Resources of CAS@home

6. Conclusions

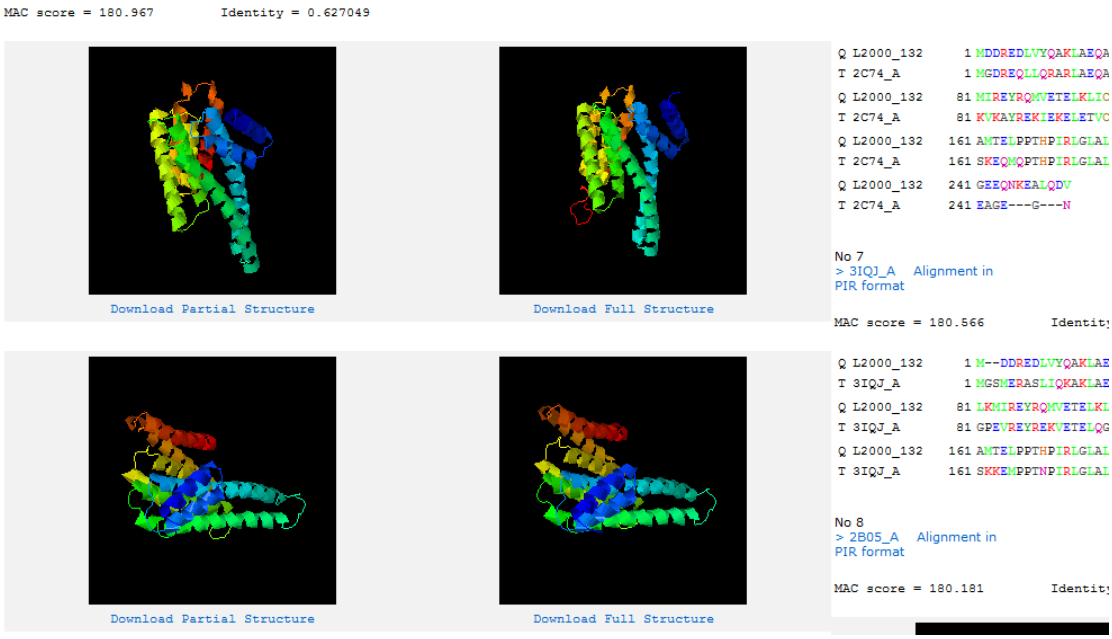

Figure 6: A Predicted Protein Structure

References

[1] Anderson D P. BOINC: A system for public-resource computing and storage [C], Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on. IEEE, 2004: 4-10.

[2] Al-Lazikani B, Jung J, Xiang Z, et al. Protein structure prediction[J]. Current opinion in chemical biology, 2001, 5(1): 51-56.

[3] Baker D, Sali A. Protein structure prediction and structural genomics [J]. Science Signalling, 2001, 294(5540): 93.

[4] BOINC, http://boinc.berkeley.edu/

[5] Anderson D P, Korpela E, Walton R. High-performance task distribution for volunteer computing [C], e-Science and Grid Computing, 2005. First International Conference on. IEEE, 2005: 8 pp.-203.

PoS(ISGC 2013)037

[6] Anderson D P, Reed K. Celebrating diversity in volunteer computing [C], System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on. IEEE, 2009: 1-8.
[7] Anderson D P. Local scheduling for volunteer computing [C], Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International. IEEE, 2007: 1-8.
[8] TreeThreader sequence submission, http://protein.ict.ac.cn/TreeThreader/
[9] TreeThreader protein structure display, http://protein.ict.ac.cn/TreeThreader/results.php?batchid=1832
[10] CAS@home, http://casathome.ihep.ac.cn
[11] CAS@home project statistics, http://casathome.ihep.ac.cn/cas_stats.html

PoS(ISGC 2013)037

