Bioinformatics Advance Access published September 23, 2010

PAR: A PARallel And Distributed Job Crusher

Francois Berenger[†]*, Camille Coti[‡] and Kam Y. J. Zhang[†]

[†]Zhang Initiative Research Unit, Advanced Science Institute, RIKEN, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan

[‡]Iowa State University, Ames, IA 50011-2251, USA

Associate Editor: Prof. Anna Tramontano

ABSTRACT

Summary: Bioinformaticians are tackling increasingly computation-intensive tasks. In the meantime, workstations are shifting towards multi-core architectures and even massively multi-core may be the norm soon. Bag-of-Tasks (BoT) applications are commonly encountered in bioinformatics. They consist of a large number of independent computation-intensive tasks. This note introduces PAR, a scalable, dynamic, parallel and distributed execution engine for Bag-of-Tasks. PAR is aimed at multi-core architectures and small clusters. Accelerations obtained thanks to PAR on two different applications are shown.

Availability: PAR is released under the GNU General Public License version three and can be freely downloaded [1].

Contact: berenger [at] riken.jp

1 INTRODUCTION

Bioinformaticians are significant high-performance computing users, in particular for simulations of biologic phenomena. On the other hand, the available hardware is getting faster but also much more parallelized (Intel publicly reported working on 80 cores prototype chips in 2007). In this context, most bioinformaticians could benefit from an easy-to-use software to harness such computing power.

The focus of this note is Bag-of-Tasks (BoT) applications execution. As the name suggests, BoT applications can be seen as a bag, filled with tasks to do, each being independent from all the others. A middle-ware for BoT applications is called a *job crusher*. It has to consist of at least a server component connected to a set of clients.

This note introduces PAR, a parallel and distributed job crusher working in *pull* mode and inspired by desktop grid platforms. Workers *join* the computation and can be added dynamically at run-time; the server delivers tasks to workers available at a given moment. PAR is actually a transposition of some concepts and features from previous distributed middle-ware to small HPC clusters and multi-core workstations.

This paper is organized as follows: Section 2 presents an overview of related projects and technologies used in bioinformatics. Section 3 presents two examples using PAR to illustrate scalability. The last section lists upcoming enhancements.

2 RELATED PROJECTS

A wide variety of tools and technologies have been used over the last two decades in bioinformatics. While PAR is a user-level tool with its own niche, it has some limitations. At the cost of a little more complexity, some of the tools listed hereafter allow fair share of resources, stronger reliability and even faster job or data throughput.

At the programming level, the Message-Passing Interface (MPI, Forum (1994)), CORBA (Object Management Group (1998)) or even MapReduce (Dean and Ghemawat (2004)) are noteworthy technology candidates.

MPI has become the *de facto* standard for programming highly parallel applications. It has been used in computational genomics (Swain *et al.* (2005)) and in molecular dynamics (Johnston *et al.* (2005); de Lomana *et al.* (2008)).

For applications following a client-server model, CORBA can be used. Handling of genome maps has successful examples (Hu *et al.* (1998), Jungfer and Rodriguez-Tomé (1998)).

For data-intensive applications, MapReduce and its open source implementation Hadoop[2] are more appropriate. They unleash operations over huge amounts of data and were used recently in sequence alignment (Sadasivam and Baktavatchalam (2010)).

However, at the application level, Desktop Grids (DG) are closer to the focus of this note. A server distributes tasks to workers located on machines that do not communicate with each other, potentially anywhere on the Internet. Condor (Litzkow *et al.* (1988)), XtremWeb (Fedak *et al.* (2001)) and BOINC (Anderson (2004)) are three platforms for highly parallel, multi-user applications. One of the best-known DG project in bioinformatics is probably Folding@home (Beberg *et al.* (2009)).

Like Hadoop and unlike most DG, PAR is designed to be used exclusively on *private* resources. PAR's ideal scale is then smaller than what DG systems usually target, but this permits a lower latency. For simplicity, PAR uses pull-driven task distribution. This removes the need for a complex software component (called a scheduler) and also allows to scale smoothly even in large, dynamic and heterogeneous environments. In addition, PAR never requires administrator privileges and is only run on-demand.

*to whom correspondence should be addressed

[1] http://git.savannah.gnu.org/cgit/par.git/plain/par.tgz

[2] http://hadoop.apache.org

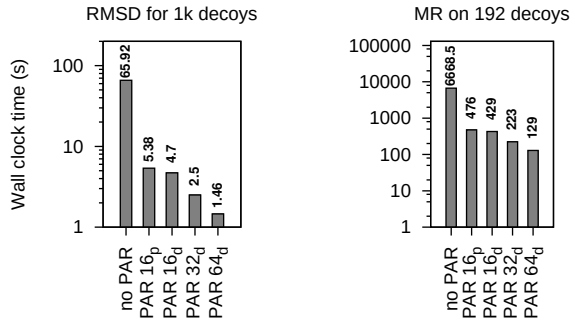

Fig. 1. Experiments accelerated by PAR (N_p: N parallel CPUs; N_d: N distributed CPUs).

3 EXAMPLE USE

The first example experiment consists of computing Alpha Carbons Root Mean-Square Deviation after optimal superposition, noted $C_\alpha RMSD_{opt}$ hereafter, on one thousand *ab initio* generated structures for the protein target 256B. Distances between proteins are computed using the software from (Zhang and Skolnick (2004)). The second experiment performs Molecular Replacement (MR), a method of solving the phase problem in X-ray crystallography using homologous structures, on a set of 192 decoys for the protein target 1m6t. We present the time elapsed with and without using PAR. PAR in parallel mode uses several cores of a given computer while the distributed mode uses distinct computers. The current implementation of PAR is known to work well with up to 16 and 64 CPUs in parallel and distributed mode respectively.

Prior to timing experiments, needed programs and data were copied to each machine by the user. During experiments, PAR was started in server mode with a list of commands to execute. Workers were started soon after the server, but could have joined the computation later if we were not interested in the shortest completion time. The Unix 'time' command was used and averaged over two trials to measure the real time spent by PAR to complete all tasks. Unlike previous job crushers, PAR server's life cycle is only tied to the application's execution time (no Unix *daemon* involved) and PAR runs only in user-space.

Results are shown in Figure 1. The first bar is the real time elapsed when not using PAR. The second bar is the time spent when using PAR in parallel mode, following bars are durations in distributed mode. On a CPU-intensive task and when using 16 CPUs, the speedup obtained by PAR can be as high as 14.01 in the parallel case and 15.54 in the distributed one. Lower performance of the parallel version is attributed to Python's problem with multi-thread applications (the Python interpreter uses a global lock mechanism shared by all threads). We can see that the application scales remarkably well. The overhead due to communications between workers and the master is very small, this allows for an effective use of the parallel hardware with minimum effort required on the user's side.

4 FURTHER DEVELOPMENTS

PAR can be used on network of Unix-like workstations. It can take advantage of a Network shared File System (NFS). However, because of poor NFS performances, data-intensive tasks should be computed on top of a Distributed File System (DFS). As DFS are still rare even within clusters, we envisage to plug in such a functionality into PAR. A prototype has been implemented but is still in experimental stage.

PAR should integrate fault-tolerance policies, in order to be used safely even with more workers over longer periods, and with minimal overhead.

Furthermore, compression could be added to speedup communications. Encryption would be similarly easy to add and would allow PAR to be used over untrusted networks.

Finally, features can be added for large-scale experiments. For example, requesting groups of jobs instead of one at a time would lower the load on the server part. Allowing PAR to run both as a server and as a client would allow it to be deployed in layers, which could be used to connect several clusters together and increase scalability. Requests and contributions from users are also considered.

ACKNOWLEDGMENTS

Funding: Our research is funded by the "Initiative Research Unit" program from RIKEN, Japan. We thank all the PAR users, especially early ones like Rojan Shrestha for providing feedback and useful feature requests. We wish to thank RIKEN, Japan, for an allocation of computing resources on the RIKEN Integrated Cluster of Clusters (RICC) system.

REFERENCES

Anderson, D. P. (2004). BOINC: A system for public-resource computing and storage. pages 4–10, Pittsburgh, PA, USA. IEEE Computer Society.

Beberg, A. L., Ensign, D. L., Jayachandran, G., Khaliq, S., and Pande, V. S. (2009). Folding@home: Lessons from eight years of volunteer distributed computing.

de Lomana, A. L. G., Gómez-Garrido, A., Sportouch, D., and Villà-Freixa, J. (2008). Optimal experimental design in the modelling of pattern formation. *ICCS'08*, **5101**, 610–619.

Dean, J. and Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters. In *OSDI'04*, Berkeley, CA, USA. USENIX Association.

Fedak, G., Germain, C., Néri, V., and Cappello, F. (2001). Xtremweb: A generic global computing system. In *CCGRID'01*, pages 582–587. IEEE Computer Society.

Forum, M. P. I. (1994). MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Department of Computer Science, University of Tennessee. Tue, 22 May 101 17:44:55 GMT.

Hu, J., Mungall, C., Nicholson, D., and Archibald, A. L. (1998). Design and implementation of a corba-based genome mapping system prototype. *Bioinformatics*, **14**(2), 112–120.

Johnston, M. A., Galván, I. F., and Villà-Freixa, J. (2005). Framework-based design of a new all-purpose molecular simulation application: The adun simulator. *Journal of Computational Chemistry*, **26**(15), 1647–1659.

Jungfer, K. and Rodriguez-Tomé, P. (1998). Mapplet: a corba-based genome map viewer. *Bioinformatics*, **14**(8), 734–738.

Litzkow, M., Livny, M., and Mutka, M. (1988). Condor - a hunter of idle workstations. In *ICDCS'88*.

Object Management Group (1998). *The Common Object Request Broker: Architecture and Specification*. Version 2.3. Object Management Group, Framingham, MA, USA.

Sadasivam, G. S. and Baktavatchalam, G. (2010). A novel approach to multiple sequence alignment using hadoop data grids. In *MDAC '10*, pages 1–7, New York, NY, USA. ACM.

Swain, M., Hunniford, T., Mandel, J. J., Palfreyman, N. M., and Dubitzky, W. (2005). Modeling gene-regulatory networks using evolutionary algorithms and distributed computing. *CCGRID'05*, **1**, 512–519.

Zhang, Y. and Skolnick, J. (2004). Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, **57**(4), 702–710.