Future Generation Computer Systems 71 (2017) 18–31

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Copernicus, a hybrid dataflow and peer-to-peer scientific computing platform for efficient large-scale ensemble sampling

Iman Pouya [a], Sander Pronk [a], Magnus Lundborg [b], Erik Lindahl [a,b,*]

[a] Swedish eScience Research Center, Department of Theoretical Physics, KTH Royal Institute of Technology, Stockholm, Sweden
[b] Department of Biochemistry and Biophysics, Science for Life Laboratory, Stockholm University, Sweden

HIGHLIGHTS

- Hybrid dataflow and peer-to-peer computing to fully automated ensemble sampling.
- The platform automatically distributes workloads and manages them resiliently
- Problems are defined as workflow by reusing existing software and scripts.
- Portability in networks where parts are behind firewalls.

ARTICLE INFO

Article history:
Received 8 December 2015
Received in revised form
18 October 2016
Accepted 5 November 2016
Available online 10 December 2016

Keywords:
Peer-too-peer
Distributed computing
Dataflow programming
Scientific computing
Job resiliency

ABSTRACT

Compute-intensive applications have gradually changed focus from massively parallel supercomputers to capacity as a resource obtained on-demand. This is particularly true for the large-scale adoption of cloud computing and MapReduce in industry, while it has been difficult for traditional high-performance computing (HPC) usage in scientific and engineering computing to exploit this type of resources. However, with the strong trend of increasing parallelism rather than faster processors, a growing number of applications target parallelism already on the algorithm level with loosely coupled approaches based on sampling and ensembles. While these cannot trivially be formulated as MapReduce, they are highly amenable to throughput computing. There are many general and powerful frameworks, but in particular for sampling-based algorithms in scientific computing there are some clear advantages from having a platform and scheduler that are highly aware of the underlying physical problem. Here, we present how these challenges are addressed with combinations of dataflow programming, peer-to-peer techniques and peer-to-peer networks in the *Copernicus* platform. This allows automation of sampling-focused workflows, task generation, dependency tracking, and not least distributing these to a diverse set of compute resources ranging from supercomputers to clouds and distributed computing (across firewalls and fragile networks). Workflows are defined from modules using existing programs, which makes them reusable without programming requirements. The system achieves resiliency by handling node failures transparently with minimal loss of computing time due to checkpointing, and a single server can manage hundreds of thousands of cores e.g. for computational chemistry applications.

1. Introduction

Computational power has evolved from a precious resource into a commodity, and new access models such as cloud computing have paved the road for large-scale computing in industry. Based on history, we can expect another order-of-magnitude increase every five years. However, this does not mean computing is no longer a bottleneck: While throughput computing works well for many classes of problems that can use the MapReduce programming model, it remains a huge challenge for most traditional high-performance computing applications in science and engineering that rely on active parallelism using message-passing interfaces and low-level parallelization. A handful of problems (for instance computational fluid dynamics) still scale well to 100,000 or more processors by increasing the resolution, but for most applications it is becoming a major concern that there is no longer sufficient inherent parallelism in the algorithm

* Corresponding author at: Swedish eScience Research Center, Department of Theoretical Physics, KTH Royal Institute of Technology, Stockholm, Sweden.
E-mail address: erik@kth.se (E. Lindahl).

http://dx.doi.org/10.1016/j.future.2016.11.004

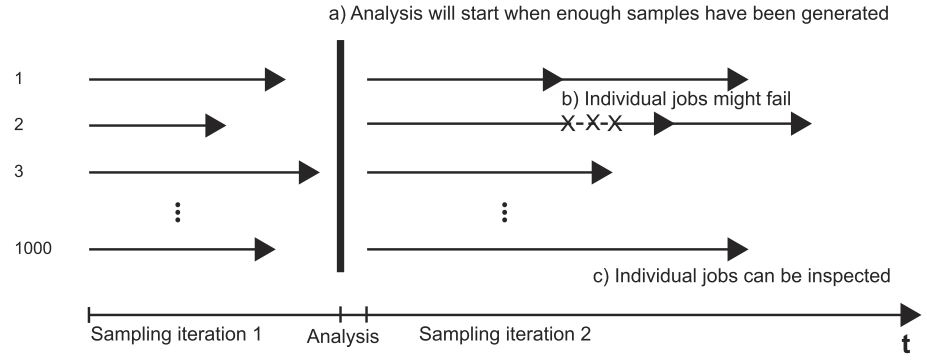

Fig. 1. Characteristics of iterative sampling methods. To generate enough samples, thousands of loosely coupled jobs can be started in parallel and analysed as soon as enough samples have been generated (a). The analysis step determines whether another round of sampling should be started or if we have converged to a solution. Some sampling problems can take days or weeks to compute and it is likely that some of the individual jobs will fail (b) during this period due to machine or user errors, and it must be possible to recover from these automatically. The input parameters and results of each job should also be transparent (c) for auditing and monitoring purposes.

to keep extending the strong scaling regime. This is particularly true for iterative simulation algorithms based on discrete time steps, for instance molecular dynamics. As the iteration times have moved into the microsecond range, strong scaling is simply limited by the network latency rather than the throughput of compute units. To overcome this impasse, there is an increasing interest in ensemble methods such as Markov state modelling [1–3], milestoning [4], strings using swarms [5,6] and metadynamics [7, 8]. The common feature of these solutions is to reformulate a single long simulation as an ensemble sampling problem that can use many short simulations to produce a single observation such as an average structure or reaction rates. This approach is not limited to molecular dynamics; many other numerical algorithms such as Monte Carlo simulations, finite element analysis, or stochastic PDEs are random in nature and can be formulated as iteratively generating samples, mixed with analysis steps that in turn generate seeds for new sampling rounds (Fig. 1).

This is a powerful way to explore states in an unknown high-dimensional parameter space, and by using adaptive algorithms to generate new seeds the total sampling can be much more efficient than relying on single long computations. By reformulating the problem into a statistical one, an entirely new layer of parallelism is exposed on the algorithm level since many loosely coupled computations can be used to generate data with synchronization intervals ranging from minutes to hours. This resolves the latency bottlenecks and makes the methods suitable e.g. for distributed computing, but the approach is equally useful for supercomputers where parallelization can be combined with tight ensemble synchronization, not to mention that it achieves close to perfect resiliency against node failures. Despite these advantages, the introduction of ensemble sampling in HPC applications has been relatively slow, for several reasons: (i) It is not until the last few years we have started to see the end of strong scaling, (ii) rewriting applications to be ensemble-focused is a large amount of work, (iii) this far, users have been required to do large parts of the statistical modelling and adaptive step manually, and (iv) managing thousands of simulations and handling analysis and new iterations quickly becomes a tedious task.

These problems can largely be solved with more automated techniques to handle the sampling. Just as many users treat the program running on the supercomputer as a black box (although they are aware of the algorithm), we believe there is great potential in completely hiding the execution of individual simulations and rather work with scientific problems and sampling algorithms as the black boxes. One successful example of this type of commoditization in computational chemistry is the Plumed [9] package that integrates with a number of different molecular dynamics packages for free energy sampling, which has enabled a number of teams to scale molecular dynamics to very large resources [10,11].

To handle a broader set of use cases, it must be possible for users to create custom automated pipelines that generate workloads on the HPC program or physical problem level and automatically distribute them to matching compute resources. Peer-to-peer (p2p) networking [12], grid computing [13–15], and dataflow programming [16] are existing concepts that all target some of these challenges, and in combination they provide very powerful solutions.

There are many existing grid and p2p computing implementations such as Condor [17,18], Gnutella [19], Boinc [20], Dirac [21], and Falkon [22]. These all aim at consolidating heterogeneous compute resources to optimally match a compute job to a resource. They give users very granular control and are very well suited for setting up a custom grid and exposing a queue for users to submit individual jobs. Similarly for dataflow programming, implementations such as Naiad [23], Flumejava [24], Millwheel [25], and Swift [26] provide domain-specific languages and libraries that give users the flexibility to create powerful and efficient implementations. As a concept, dataflow programming has mostly been orthogonal to grid or p2p computing, but systems such as Pegasus [27], Kepler [28], Triana [29], and Fireworks [30] have emerged that combine them. MapReduce is also a beautiful example of the power in combining concepts [31]; this was originally a programming model, but today it is more coupled to distributed and cloud computing as it has been extended with a problem-aware engine that handles workload distribution automatically, and it works very well for large-scale data analysis. While the above-mentioned dataflow implementations are very general, some of them can also come with a rather steep learning curve (especially for non-programmers), and porting an existing program that relies e.g. on simulations can be a relatively large undertaking since the entire algorithmic approach has to fit the specific programming model. In particular, for our own field of scientific computing, it is very common to base work on large, old, and complex (parallel) programs, which are executed either on supercomputers or in distributed computing environments to generate statistical data. If these codes could be combined into automated workflows it would both extend them with distributed computing capabilities and enable access to a much wider range of throughput-focused computational resources, without much extra work.

The wishlist for ensemble sampling would be to combine the features of existing platforms with a system that can manage a dynamic workflow behaviour, allow users to exploratively alter input values, inspect input/output values during runtime, and leave individual job management to be handled automatically. To lower the barrier of entry it should be possible to reuse existing