Future Generation Computer Systems 59 (2016) 114–124

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Application skeletons: Construction and use in eScience

Daniel S. Katz[a,*], Andre Merzky[b], Zhao Zhang[c], Shantenu Jha[b]

[a] Computation Institute, University of Chicago & Argonne National Laboratory, Chicago, IL, USA
[b] RADICAL Laboratory, Rutgers University, New Brunswick, NJ, USA
[c] AMPLab, University of California, Berkeley, CA, USA

HIGHLIGHTS

- Skeleton applications represent the key parameters of parallel and distributed eScience applications.
- Skeleton applications are easy-to-program, easy-to-build, and easy-to-use, and easy-to-share.
- Skeleton applications have similar performance to the real applications on which they are based.
- Skeletons application are built from an open source: https://github.com/applicationskeleton/Skeleton.
- Skeleton applications can be used to demonstrate system optimizations.

ARTICLE INFO

Article history:
Received 24 March 2015
Received in revised form
18 September 2015
Accepted 3 October 2015
Available online 22 October 2015

Keywords:
Computational science
Data science
Application modeling
System modeling
Performance modeling
Parallel and distributed systems

ABSTRACT

Computer scientists who work on tools and systems to support eScience (a variety of parallel and distributed) applications usually use actual applications to prove that their systems will benefit science and engineering (e.g., improve application performance). Accessing and building the applications and necessary data sets can be difficult because of policy or technical issues, and it can be difficult to modify the characteristics of the applications to understand corner cases in the system design. In this paper, we present the Application Skeleton, a simple yet powerful tool to build synthetic applications that represent real applications, with runtime and I/O close to those of the real applications. This allows computer scientists to focus on the system they are building; they can work with the simpler skeleton applications and be sure that their work will also be applicable to the real applications. In addition, skeleton applications support simple reproducible system experiments since they are represented by a compact set of parameters.

Our Application Skeleton tool (available as open source at https://github.com/applicationskeleton/Skeleton) currently can create easy-to-access, easy-to-build, and easy-to-run bag-of-task, (iterative) map-reduce, and (iterative) multistage workflow applications. The tasks can be serial, parallel, or a mix of both. The parameters to represent the tasks can either be discovered through a manual profiling of the applications or through an automated method. We select three representative applications (Montage, BLAST, CyberShake Postprocessing), then describe and generate skeleton applications for each. We show that the skeleton applications have identical (or close) performance to that of the real applications. We then show examples of using skeleton applications to verify system optimizations such as data caching, I/O tuning, and task scheduling, as well as the system resilience mechanism, in some cases modifying the skeleton applications to emphasize some characteristic, and thus show that using skeleton applications simplifies the process of designing, implementing, and testing these optimizations.

Published by Elsevier B.V.

1. Introduction

Computer scientists who build tools and systems (programming languages, runtime systems, file systems, workflow systems, etc.) to enable eScience often have to work on real scientific applications to prove the effectiveness of the system. Accessing and building the real applications can be time consuming or sometimes infeasible for one or more of the following reasons:

- Some applications (source) are privately accessible.
- Some data are difficult to access.
- Some applications use legacy code and are dependent on out-of-date libraries.

* Corresponding author.
E-mail addresses: d.katz@ieee.org (D.S. Katz), andre@merzky.net (A. Merzky), zhaozhang@eecs.berkeley.edu (Z. Zhang), shantenu.jha@rutgers.edu (S. Jha).

http://dx.doi.org/10.1016/j.future.2015.10.001
0167-739X/Published by Elsevier B.V.

- Some applications are hard to understand because they implicitly assume domain knowledge.

In addition, real applications may be difficult to scale or modify in order to demonstrate system trends and characteristics. Our work is partially motivated by the state of distributed applications. We previously highlighted the challenges of developing distributed applications, showing that the lack of development abstractions and the complexity of deployment were two important barriers [1,2].

Our Application Skeletons idea was created in the AIMES project, whose goal is to explore the role of abstractions and integrated middleware to support eScience at extreme scales. AIMES is co-designing middleware from an application and infrastructure perspective. Thus, it requires applications with various characteristics for better application coverage. We have previously encountered many problems when accessing real applications and when trying to distribute applications and data as test cases to other researchers. Application Skeletons are intended to overcome these issues.

We previously presented [3] the Skeleton idea of working around such issues by quickly and easily producing a synthetic distributed application that is executable in a distributed environment, for example, grids, clusters, and clouds, and then showed [4] improvements in reducing the gap between skeleton and real application performance, as well as showing Skeleton applications could be used to simplify understanding and demonstrating system optimizations, such as AMFORA [5].

The Application Skeleton tool takes as input an application description file composed in a top-down approach: an application was described as a number of stages, and each stage had a number of tasks. Users specify tasks at the stage level by articulating the number of tasks, task lengths, and input/output file sizes. Applications can be composed of serial tasks, parallel tasks, or a mix of both. The task is implemented as a versatile C program, and the compiled executable can be serial or parallel dependent on how it is compiled. Users can specify a task's read/write buffer size, since such buffers are often used in real application code. The Skeleton task can mimic real application tasks' interleaving behavior for reads, writes, and computation. The tasks are C programs compiled to static executables, so they can run on supercomputers with an OS that does not have fork()/exec() support, such as the IBM Blue Gene/Q. Some of the task parameters, such as task lengths and input file sizes, can be described as a statistical distribution. The task implementation is based on the UNIX/Linux sleep and dd programs, controlling the CPU time and I/O, respectively.

This Skeleton implementation can generate bag-of-task, (iterative) map-reduce, and (iterative) multistage workflow applications. The skeleton applications are executable with common distributed computing middleware, Swift [6–8] and Pegasus [9,10], as well as the ubiquitous UNIX shell on a single site (a local cluster with a shared file system), and the skeleton set of tasks can also be output as a generic JSON object that can be used by other systems, such as in our case, by our AIMES middleware.

Skeleton parameters can be discovered by either by manually or automatically profiling the application. Once the real application is represented by a Skeleton description file, it can easily be distributed and reused. We measured the performance error between the skeleton application against three real applications (Montage [11,12], BLAST [13], and CyberShake PostProcessing [14]) on 64 processors of a BG/P supercomputer with per stage and total errors of between 1 and 3%. And changing the parameters makes it easy to study how the system responds to different application characteristics, specifically data caching, task scheduling, I/O tuning, and resilience mechanism.

This paper summarizes our previous work [3,4] and adds further work toward automated, system independent application profiling. Specifically, almost all of Section 4 is new, and Section 2 (related work) has been expanded to provide better context for our work; as such, it also has been moved to earlier in the paper. Sections 3 and 5 were initially shortened to focus on key points and then modified to include automated profiling, then further modified as part of the review process. The introduction (Section 1) and conclusions (Section 7) have been reworked for this paper but used the introductions and conclusions of the previous papers as starting points. Section 6 has just been condensed from [4].

The contributions of this work include the following:

- An application abstraction that gives users good expressiveness and ease of programming to capture the key performance elements of distributed applications.
- A versatile Skeleton task implementation that is configurable (serial or parallel, number of processes, read/write buffer size, input/output files, interleaving options).
- A comparison of two methods for the estimation of Skeleton parameters: manual, system specific application profiling versus automated, system independent profiling.
- An interoperable Skeleton implementation that works with mainstream workflow frameworks and systems (Swift, Pegasus, and Shell), and allow general output for other systems.
- The usage of Skeleton applications to simplify system optimization implementation and highlight their impacts.

The rest of the paper is organized as follows: Section 2 discusses the idea of application modeling, including related work. Section 3 introduces the design of the Application Skeleton tool and the tradeoffs we made during the process. In Section 4, we discuss how Skeleton parameters can be determined. In Section 5, we select three representative applications, and compare the Skeleton application performance against the real application performance. In Section 6, we show how application skeletons can help eScience infrastructure developers. Conclusions are drawn in Section 7.

2. Modeling background and related work

Researchers have been using application replacements of various types (e.g., kernels, benchmarks, reduced applications, miniapps, traces) for experiments for a long time, and some of those replacements have been and are certainly tunable in one way or another. These application replacements are used for a variety of purposes, including:

- They are easier to build than the actual application.
- They can be reused across different system to compare those systems.
- They run faster than the actual application.
- They can be shared with collaborators who do not have access to the application or the data.

However, we believe that the idea of using application skeletons, particularly for overall system performance, is relatively novel. The distinction in our work is that the Skeletons provide a systematic application replacement capability, which both preserves the significant part of the application's behavior and is tunable across a range of diverse applications. Because of the problem space is pervasive but attempts to solve it systematically are rare, particularly for distributed applications, the related work we discuss here are examples of some of the different types of application replacements that have been used.

Examples of benchmarking/analysis work in a parallel (not distributed) context include the NAS Parallel Benchmarks [15], and Berkeley Dwarfs [16] (also called 'motifs') and the related OpenDwarfs [17]. The NAS benchmarks include both kernels and pseudo-applications (an example of reduced applications), intended for use on parallel (not distributed) computers. These