Validation Framework for Multiprocessor and Distributed Scheduling Algorithms

Hamilton Turner
Dept. of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA, USA
hamiltont@vt.edu

Abstract—This work examines the challenge arising from the practical use of Deployment Optimization Algorithms, and primarily focuses on the need for validation of potentially beneficial algorithms. Deployment Optimization Algorithms are appropriate for optimizing many distributed and multicore systems, but even well-known algorithms can have unclear results when used in a production system due to the tendency of researchers to use a simplified model of computation when evaluating algorithms. Unexpected challenges, such as bus contention, are frequently not considered when evaluating Deployment Optimization Algorithms, which can lead to unexpected results in production environments. This paper presents DOVE, a framework for validating Deployment Optimization Algorithms on real systems.

I. INTRODUCTION

Emerging Trends and Challenges. Distributed and multicore systems are critical in modern computing. These systems are used for problem domains such as graphics computing and digital signal processing. While the theoretical processing power of these systems is quite large, there are a number of challenges to effectively utilizing all cores. In the general case, and even in many cases with relaxed assumptions, obtaining optimal mappings has been shown to be NP-Hard [1], [2]. The motivation for this challenge has increased substantially as computer processors have become increasingly multicore.

Multicore Deployment Optimization (MCDO) formalizes the optimization challenge, which aims to provide a mapping of software tasks onto hardware processors in such a way that an objective function, typically the makespan or overall execution time of all jobs, is minimized. Each software task has pre-established dependencies, as shown in Figure 1, that restrict a task from running until all of its predecessor tasks have completed. If a task \( A \) and its dependency task \( A' \) are executed on different processing cores \( P \) and \( P' \), then a message must be routed from \( P' \) to \( P \) upon completion of \( A' \). The time required to route this message is dependent upon the two processing cores in question, as neighboring cores can communicate more quickly than remote cores. Moreover, each processing core is considered heterogeneous, so some cores will execute software tasks rapidly while others will execute tasks slowly.

While multiple algorithms have been proposed to solve this problem, practical use of these algorithms requires both comparison on of these algorithms to each other [3] and effective validation. As stated by the United States Department of Defense, validation is “The process of determining the degree to which a model or simulation and its associated data are an accurate representation of the real world from the perspective of the intended uses of the model. [4]” Deployment Optimization Algorithms accept as input a model of production system’s hardware and software, they try to minimize an objective function such as power consumption, and output the a deployment plan that will result in the real system having similar improved performance on this objective function.

Open Problem ⇒ Validation of Deployment Optimization Algorithms.

While Deployment Optimization is an active research topic, there are few real-world case studies of using the algorithms. Moreover, system designers interested in using a Deployment Optimization Algorithms cannot be guaranteed that the performance of that algorithm on their production system will equal the performance shown in the literature. Deployment Optimization Algorithms performance is not measured on real systems, but is instead is measured using an approximation of expected runtime that is composed of expected computation and routing delays. This approximation does not take into
account issues such as memory size limits, routing contention, or disk I/O delays. Such issues arise can cause significant variation between the expected and actual performance of a Deployment Optimization Algorithms.

**Solution Approach ⇒ Automation of Deployment Optimization Algorithms validation.** To address the challenge of validation of Deployment Optimization Algorithms s we have create the DOVE framework, which automates the process of ensuring that the reported improvements in the objective function are consistent with the profiled improvements seen in a physical system. A physical hardware system is profiled to provide computation and routing delays to a Deployment Optimization Algorithms, and the full set of \{solution, objective function value\} pairs is then executed and profiled on a physical system.

This paper provides the following contributions to the study of Deployment Optimization Algorithms:

- DOVE, a framework for automating the validation of Deployment Optimization Algorithms
- Extensive discussion on validation of Deployment Optimization Algorithms

The remainder of this paper is organized as follows: Section II discusses the challenges that arise during validation of distributed and multicore Deployment Optimization Algorithms; Section III outlines DOVE, which is our solution to validation of Deployment Optimization Algorithms; Section IV presents work similar to our validation of Deployment Optimization Algorithms; and Section V presents concluding remarks and lessons learned.

**II. CHALLENGES OF VALIDATING DEPLOYMENT OPTIMIZATION ALGORITHMS**

Deployment Optimization Algorithms are an active area of research precisely because they have the potential to cause substantial performance improvements without incurring additional physical hardware costs. However, there are a number of challenges to practical use of a Deployment Optimization Algorithms, such as the inability to know the precise effects of applying a Deployment Optimization Algorithms to an existing production environment. This section details some of the fundamental challenges to validating that a Deployment Optimization Algorithms algorithm can actually confer real-world benefits equal to the predicted improvements.

**A. Challenge 1: Determining if Deployment Optimization is Improving Production System Metrics is Hard**

Deployment Optimization Algorithms use an objective function to choose which deployment plan is most appropriate. However, there is no guarantee that the value of the objective function calculated by the Deployment Optimization Algorithms will equal the value of the objective function when executing the real system. There is little confidence that the objective function on the real system will equal the objective function value predicted by the Deployment Optimization Algorithms, due to the Deployment Optimization Algorithms operating on a simplified model of the real system that does not considering all complications, such as bus contention.

While Deployment Optimization Algorithms have been shown multiple times to have beneficial effects [3], the exact amount of the effect can vary widely between each system execution.

**B. Challenge 2: Identifying Reasonable Computation Limits for Optimization is Difficult**

Deployment Optimization Algorithms are typically iterative algorithms, but it is unclear how many iterations are needed to find the best deployment plan without wasting compute time on the optimization. Many Deployment Optimization Algorithms include randomness, which implies that if they are allowed to run long enough they will find the optimal solution, although the algorithm user has no knowledge that this is in fact the optimal solution– extra computation time can be wasted trying to find a better solution! It is hard to predict a reasonable termination condition for any Deployment Optimization Algorithms. Moreover, algorithm performance is drastically problem dependent, and a reasonable termination condition on one problem may perform poorly on another.

**III. THE DEPLOYMENT OPTIMIZATION VALIDATION ENGINE (DOVE)**

DOVE enables automated validation of deployment optimization algorithms. Figure 2 shows the major components of DOVE, which are covered in detail in Section III. To validate a Deployment Optimization Algorithms, DOVE requires a physical hardware system and a software model, which contains information about data flow and component execution times for the software system being deployed. As shown in Figure 2, the hardware system is first profiled by DOVE to build an accurate description of available computation units and the routing delays between them. The model of production system software is obtained from a file in the Standard Task Graph (STG) format, which includes a directed acyclic graph of task dependency and exact execution times for each component.

**A. Profiler.**

The profiler tool builds an XML file describing the hardware system that will later be used for validation of a Deployment Optimization Algorithms. First, the Portable Hardware Locality tool provided by Open MPI is used to gather details on the available hardware components e.g. processors, cores, and hardware threads. Next, a latency calculation is performed between all hardware components pairs at the same level (i.e. core-core, processor-processor, etc). Most Deployment Optimization Algorithms s reviewed used a single number to represent the routing delay between processing units, and therefore DOVE’s profiler runs hundreds of thousands of repetitions and returns the average routing time. Final profiler output is shown in Listing 1.

```xml
<nodes>
  <node ip="10.0.2.4" id="0" pindex="1">
    <socket id="1" pindex="1" speed="2.4GHz">
      <core id="2" pindex="0">
        <pu id="3" pindex="0"/>
        <pu id="4" pindex="12"/>
        ... etc
```
B. STG Software Model.

Research in software profiling enables translating real software systems into representative formats such as STG [5], [6]. Using an intermediate format instead of using running real system code has multiple advantages: 1) by generating deterministic executable code from the software model, any variation from expectation can be attributed to errors in the hardware model, 2) any complexities with building, installing, or running a production environment can be hidden from DOVE. 3) it becomes simpler to generate testcases based on common software patterns, as evidenced by [3], [7]. However, this also implies that any performance output metrics generated by DOVE cannot be assumed to be the exact expected behavior on a production system. If the STG software model is similar to the real system, then the metrics may be the quite close, but there are some software systems, such as web services, where the resource requirements are dynamic and therefore cannot be effectively modeled using the STG format. Future work on DOVE will explore alternative methods of modeling software.

C. Optimization Algorithm Modifications.

DOVE is designed to require only minimal modifications to the actual optimization algorithm. All algorithms are assumed to use an optimization model similar to the one described in Section I, and DOVE provides functions to enable any optimization to be validated. Namely, request hardware components (i.e. allocate 12 cores) for use in a validation, get routing delays between hardware components, and store a deployment plan plus the associated model metrics. Typical inclusion would be about 10 lines of additional code, as shown in Listing 2.

D. Executable System Generator.

The generator included in DOVE combines a) the deployment plan output by the optimization algorithm b) the system.xml file used to find the machine-specific identifiers of
all hardware components, and c) the software model. These three components are combined into an executable and a set of rankfiles (e.g., configuration files) that can be used with Open MPI to execute the software model using the deployment plans produced by the optimization. Listing 4 shows an example of the executable software code generated for Open MPI. Open MPI is configured to wait on all predecessor tasks in an asynchronous manner. Then, a busy wait loop is executed for the amount of time equal to the task’s computation. Finally, all successor tasks are notified asynchronously. In Listing 4, debug output is enabled, but production code for profiling would not have the print statements.

```cpp
case 3: {
cout << "3: Awake" << endl;
mpi::request req[2];
req[0] = world irecv (1, 0);
req[1] = world irecv (2, 0);
mpi::wait_all (req, req + 2);
cout << "3: Recv all predecessors" << endl;
timespec start, end;
cout << "3: Started compute" << endl;
clock_gettime (CLOCK_MONOTONIC, &start);
do
    clock_gettime (CLOCK_MONOTONIC, &end);
while (diff (start, end).tv_nsec < 6000):
cout << "3: Finished compute" << endl;
mpi::request sreq[1];
sreq[0] = world isend (4, 0);
mpi::wait_all (sreq, sreq + 1);
cout << "3: Notified successors" << endl;
}
```

Listing 4. Sample of software implementation generated by DOVE

E. Implementation runner.

The implementation runner loops over each of the solutions output by the Deployment Optimization Algorithms, and times the execution of that solution. To avoid overly long profiling, the kbest algorithm is used to terminate profiling of one solution once the top k scores have converged to a threshold range, such as 500 ns. After profiling each deployment, the implementation runner updates the deployment XML file to include the total time required to execute each department. Each deployment node inside of the deployment XML file will now include a new node "<metric name='time' value='7.8988e05' unit='seconds'/>".

F. Analyzer.

The name analyzer performs a hypothesis test with $H_0$ equaling “The objective function of the modelled system will not correlate with the objective function of the profiled system.” Future works on this component will examine detecting the number of iterations before the iterative improvements drops below a threshold. Additionally, future works will consider adding additional hardware metrics and trying to detect correlations between these metrics and poor solution performance, which may be useful for assisting algorithm designers in adding critical details to their models.

IV. RELATED WORK

Validation of Timing Constraints. Ha and Liu present work on validating timing constraints when jobs are independent and their execution time is only roughly known a priori [8]. Similar work has been done to validate timing constraints when scheduling periodic recurring tasks.

Comparison Using Algorithm Metrics [3] present a comprehensive comparison of multiple ‘Directed Acyclic Graph scheduling’ algorithms, using as a basis for comparison metrics such as the shortest schedule length (e.g. makespan) achieved, the number of processors used, the algorithm running time, and the range and frequency of schedule lengths. The comparison did consider that algorithms are frequently created to operate not on a real system, but on a model of multiprocessor scheduling that was similar to a real system. However, all solution metrics were calculated by ‘running’ scheduling plans on the model of multiprocessor scheduling instead of a real system. Therefore, if the model did not include potentially important effects, such as network contention, then metrics such as total makespan cannot be considered accurate for a real system. Our work extends this by examining the result of using algorithm solutions on a real system instead of on a system model.

V. CONCLUDING REMARKS & LESSONS LEARNED

This paper examined the challenge of validating Deployment Optimization Algorithms, or of ensuring that Deployment Optimization Algorithms actually resulted in improved performance metrics on real systems. Section II introduced some of the primary challenges to the practical use of Deployment Optimization Algorithms, including the difficulty of validating the algorithms and the unknown benefits of increased optimization time.

This paper addresses these concerns by automating one validation procedure for Deployment Optimization Algorithms. The chosen procedure is encapsulated in DOVE, and is shown to be simple to integrate into an existing Deployment Optimization Algorithms.

REFERENCES