How should you structure your hierarchical scheduler?

Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien
École Normale Supérieure de Lyon, France
{Pushpinder-Kaur.Chouhan,Holly.Dail,Eddy.Caron,Frederic.Vivien}@ens-lyon.fr

Abstract— **In this paper we study how distributed scheduling systems can be designed most effectively; we focus on the problem of selecting an optimal arrangement of schedulers, or a deployment, for hierarchically organized systems. We show that the optimal deployment is a complete spanning d-ary tree; this result conforms with results from the scheduling literature. More importantly, we present an approach for determining the optimal degree d for the tree. We test our approach using DIET, a network-enabled server system that uses hierarchical schedulers. Finally, we demonstrate that our approach selects deployments that are near-optimal in practice.**

I. Introduction

Efficient, scalable schedulers are key middleware components needed for users to make effective use of large collections of machines. A scheduling service matches user requests with available resources and may also negotiate amongst competing users and keep track of the status and expected performance of resources. However, little is known about how to find the best distributed arrangement, or *deployment*, of schedulers. Questions such as “which resources should be used?”, “how many resources should be used?”, “what arrangement should be used”, and “should the fastest and best-connected resource be used for a scheduler or as a computational resource?” remain difficult to answer.

Here we target a smaller sub-problem: “what is the optimal hierarchical deployment of schedulers on a cluster with hundreds to thousands of nodes?”.

II. Platform deployment planning

Software system architecture - We consider a service-provider software system composed of three types of elements: a set of client nodes $\mathbb{C}$ that require computations, a set of server nodes $\mathbb{S}$ that are providers of computations, and a set of agent nodes $\mathbb{A}$ that provide coordination of client requests with service offerings via service localization, scheduling, and persistent data management. We consider only hierarchical arrangements of agents composed of a single top-level root agent and any number of agents arranged in a tree below the root agent. Server nodes are leaves of the tree, but may be attached to any agent in the hierarchy, even if that agent also has children that are agents.

Request definition - Clients use a 2-phase process to interact with a deployed hierarchy: they submit a scheduling request to the agents to find a suitable server in the hierarchy (the scheduling phase), and then they submit a service request (job) directly to the server (the service phase). A *completed request* is one that has completed both the scheduling and service request phases and for which a response has been returned to the client.

Resource architecture - The target resource architectural framework is represented by a weighted graph $G = (\mathbb{V}, \mathbb{E}, w, B)$. Each vertex v in the set of vertices $\mathbb{V}$ represents a computing resource with computing power w in MFlop/second. Each edge e in the set of edges $\mathbb{E}$ represents a resource link between two resources with edge cost B given by the bandwidth between the two nodes in Mb/second.

Deployment assumptions - We consider that at the time of deployment we do not know the client locations or the characteristics of the client resources. Thus clients are not considered in the deployment process and, in particular, we assume that the set of computational resources used by clients is disjoint from $\mathbb{V}$. A valid deployment will always include at least the root-level agent and one server. Each node $v \in \mathbb{V}$ can be assigned either as a server or an agent or left idle. Thus with $|\mathbb{A}|$ agents, $|\mathbb{S}|$ servers, and $|\mathbb{V}|$ total resources, $|\mathbb{A}|+|\mathbb{S}| \leqslant |\mathbb{V}|$.

Optimal deployment - Our objective is to *find an optimal deployment of agents and servers for a set of resources* $\mathbb{V}$. We consider an *optimal deployment* to be a deployment that provides the maximum throughput ρ of completed requests per second.

We define the scheduling request throughput in requests per second, ρ_{sched}, as the rate at which requests are processed by agents and servers in the scheduling phase. Likewise, we define the service throughput in requests per second, $\rho_{service}$, as the rate at which the servers can produce the services required by the clients. The following lemmas lead to a proof of an optimal deployment shape of the platform. The proofs of these lemmas can be found in [1].

Lemma 1. *The completed request throughput ρ of a deployment is given by the minimum of the scheduling request throughput ρ_{sched} and the service request throughput $\rho_{service}$.*

The *degree* of an agent is the number of children directly attached to it, regardless of whether the children are servers or agents.

Lemma 2. *The scheduling throughput ρ_{sched} is limited by the throughput of the agent with the highest degree.*

Lemma 3. *The service request throughput $\rho_{service}$ increases as the number of servers included in a deployment increases.*

DGEMM Size	Nodes $\|\mathbb{V}\|$	Optimal Degree	Model selected Degree	Our model Performance	Star Performance	Tri-ary Performance
10	21	1	1	100.0%	22.4%	50.5%
100	25	2	2	100.0%	84.4%	84.6%
200	45	3	8	87.1%	40.0%	100.0%
310	45	15	22	98.5%	73.8%	74.0%
1000	21	20	20	100.0%	100.0%	65.3%

TABLE I
A SUMMARY OF THE PERCENTAGE OF OPTIMAL ACHIEVED BY DIFFERENT TREES DEPLOYMENT.

Definition 1. *A **C**omplete **S**panning **d**-ary tree (CSD tree) is a tree that is both a complete d-ary tree and a spanning tree.*

Definition 2. *A $dMax$ set is the set of all trees that can be built using $|\mathbb{V}|$ resources and for which the maximum degree is equal to $dMax$.*

Theorem 1. *In a $dMax$ set, all $dMax$ CSD trees have optimal throughput.*

Theorem 2. *A complete spanning d-ary tree with degree $d \in [1, |\mathbb{V}| - 1]$ that maximizes the minimum of the scheduling request and service request throughputs is an optimal deployment.*

III. Experimental results

DIET [2] provides a good test case for our approach; however, we need performance models for the scheduling and service phases of DIET before applying our approach. The throughput of the scheduling phase, ρ_{sched}, in requests per second is given by the minimum of the throughput provided by servers and agents for prediction. The throughput of the service phase, $\rho_{service}$, in requests per second is given by the throughput provided by the servers while taking into account that the servers spend some time doing predictions.

We tested two operating models: the serial model and the parallel model. In the serial model, a computing resource has no capacity for parallelism: it can either send a message, receive a message, or compute. In the parallel model, a computing resource can send messages, receive messages, and do computation in parallel. For both models we only assume a single port-level: messages must be sent serially and they must be received serially.

Fig. 1 shows predicted and measured throughput for star-based DIET deployments using different numbers of servers in the star and assuming a network bandwidth of 190 MB/s. The nearly linear increase of throughput with the number of children (agent degree) shows that throughput in these tests was server-limited, rather than agent-limited, and that our model is able to accurately predict performance under these conditions.

Next we tested the effectiveness of our approach in selecting a good deployment. We used two similar clusters for these tests: a 55-node cluster at Lyon and a 140-node cluster at Sophia in France. For each experiment, we select a cluster, define the total number of resources available, and pick a DGEMM problem size. We then used our deployment approach

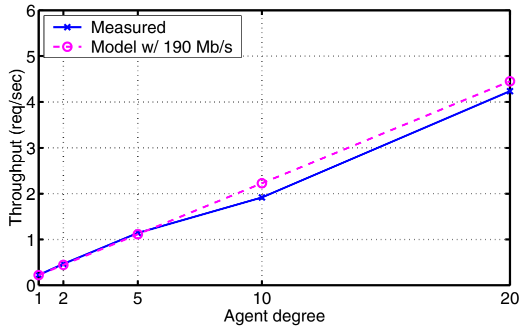

Fig. 1. Measured and predicted platform throughput for DGEMM size 1000; predictions are shown for the serial model with bandwidth 190 Mb/s.

to select a good deployment and tested throughput using the selected deployment and several other reasonable deployments. Table I summarizes the results of these experiments by reporting the percentage of the optimal throughput achieved for the tree selected by our model, the star, and the tri-ary tree. Problem size 10 represents the usage of the model in clearly server-limited conditions, while problem size 1000 represents clearly server-limited conditions.

IV. Conclusion and future Work

This paper showed that a Complete Spanning d-ary trees provide optimal hierarchical middleware deployments for homogeneous resource platforms. Our approach determines how many nodes should be used and in what hierarchical organization with the goal of maximizing steady-state throughput. We presented experiments validating the DIET throughput performance models and demonstrating that our approach can effectively build a tree for deployment which is nearly optimal and which performs significantly better than other reasonable deployments. This article provides only the initial step for automatic middleware deployment planning. Our final goal is to develop deployment planning and re-deployment algorithms for middleware on heterogeneous clusters and grids.

References

[1] E. Caron, P.K. Chouhan, H. Dail, and F. Vivien. Automatic middleware deployment planning on clusters. Research report 2005-50, Laboratoire de l'Informatique du Parall´elisme (LIP), 2005. Revised version of LIP Research Report 2005-26.
[2] E. Caron and F. Desprez. DIET: A scalable toolbox to build network enabled servers on the Grid. *International Journal of High Performance Computing Applications*, 2006.