IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society
Vol. 5, No. 10; October 2004

Profile-Driven Component Placement for Cluster-Based Online Services

Christopher Stewart, *University of Rochester*
Kai Shen, *University of Rochester*
Sandhya Dwarkadas, *University of Rochester*
Michael L. Scott , *University of Rochester*
Jian Yin, *IBM Research*

Software component placement can be a significant challenge for large-scale online services. Profile-driven tools can identify placements that achieve near-optimal overall throughput.

Online services are most often deployed on clusters[1,2] of commodity machines to achieve high availability, incremental scalability, and cost effectiveness in the face of rapid service evolution and increasing user demand. Their software architecture typically comprises many components. Some components reflect intentionally modular design; others were developed independently and subsequently assembled into a larger application—for example, to handle data from independent sources. A typical service might contain components responsible for data management, business logic, and presentation of results in HTML or XML.

Placement of these components on cluster nodes is challenging for three main reasons. First, substantial heterogeneity can exist both in component resource needs and in available resources at different nodes. Second, maintaining reasonable quality of service (for example, response time) is imperative for interactive network clients. Third, the ideal placement might be a function of not only static application characteristics but also various runtime factors, including bursty user demand, machine failures, and system upgrades. Our goal is to develop the software infrastructure needed for efficient dynamic component placement in cluster-based online services.

Our basic approach (see Figure 1) is to build per-component resource consumption profiles (or *componentprofiles*, for short) as a function of input workload characteristics. The resources we currently consider are CPU, network bandwidth, and memory usage, each of which we characterize in terms of average and peak resource requirements. Our approach then determines component placement on the basis of the profiles, available system resources, and runtime workload characteristics. Our approach can make placement decisions either at a centralized executive server or in a fully distributed fashion. It can also make decisions dynamically for runtime component migration by monitoring the input workload characteristics. As far as we know, our approach is the only one that employs a parameterization based on input workload characteristics for cluster-based online services. (For other placement

approaches, see the "Research on Distributed-Component Placement" sidebar.) This enables inexpensive monitoring and accurate prediction based on the component profiles.

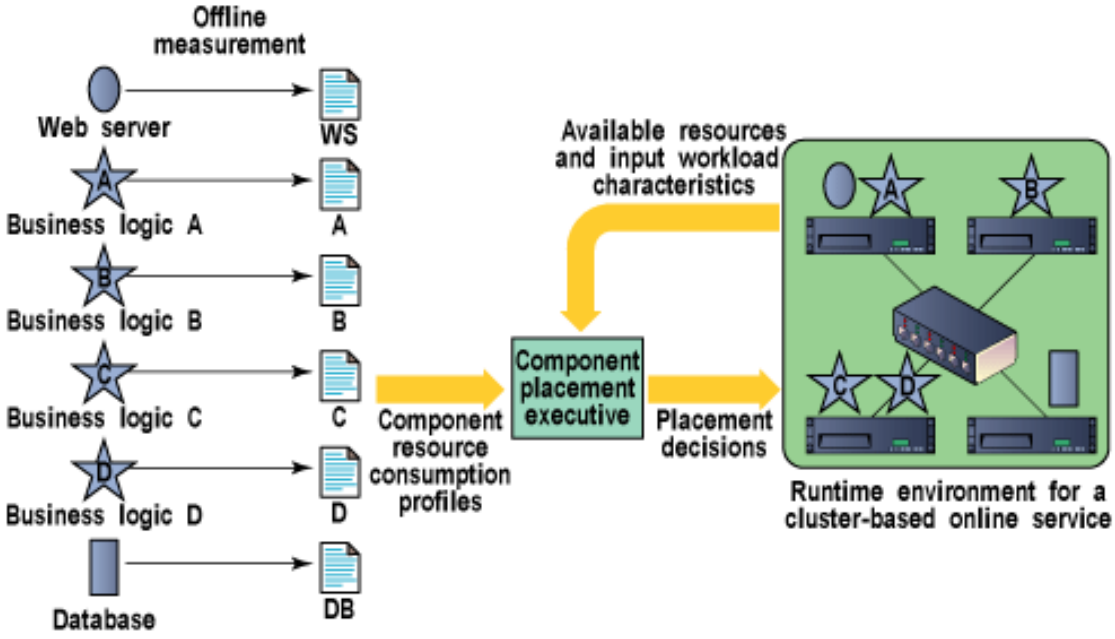

Figure 1. Profile-driven component placement.

Design framework

Our component placement system has three main features. First, it offers offline measurement and modeling mechanisms for building the component profiles. Second, it offers profile-driven performance projection and an automatic component placement strategy optimized for high system throughput. Finally, as we mentioned before, it allows for runtime workload monitoring and dynamic component migration.

Building component profiles

For the CPU and network bandwidth, an average rate r and a peak rate f captures each resource's consumption specification. We measure and accumulate resource consumption statistics at periodic intervals. The peak rate is the maximum or a high (for example, 90) percentile value of such statistics. The average and peak resource needs let us estimate upper and lower bounds on throughput.

Regarding memory usage, variability in available memory size often severely affects application performance. In particular, a memory deficit during one time interval can't be compensated simply by a memory surplus in the next time interval. So, we use only the maximum requirement f_{mem} for specifying memory requirements in the component profile.

To parameterize the input workload specifications in the profiles, we use an average request arrival rate $l_{workload}$ and other workload characteristics (denoted by $d_{workload}$) such as the composition of different request types (or *request mix*). The request mix is relevant because different request types might not consume the same resources. For example, a workload consisting entirely of Web page retrievals and one consisting entirely of new-user registrations consume very different resources.

So, the profile for a distributed application component specifies the following mapping $f()$:

$f(l_{workload}, d_{workload})$ ® $(r_{cpu}, f_{cpu}, r_{network}, f_{network}, f_{mem})$

Many functional relationships in $f()$ will likely take simple, expressible forms. For instance, $r_{cpu}, f_{cpu}, r_{network}$, and $f_{network}$ will likely be linear in $l_{workload}$.

Several techniques are available to measure system resource utilization by application components under various workloads. One such technique uses OS-provided interfaces to acquire resource consumption statistics. For instance, the SYSSTAT toolkit (see http://perso.wanadoo.fr/sebastien.godard) provides CPU, memory, and I/O resource consumption statistics through access to the Linux `/proc` interface. One drawback of such an approach is that the measurement accuracy is limited by the frequency of statistics reporting from the OS. Another technique, represented by the Linux Trace Toolkit,[3] directly instruments the OS kernel to report resource consumption statistics. LTT can provide accurate system statistics but requires significant kernel changes.

For information on other profiling research, see the sidebar "Research on Application-Resource-Consumption Profiling."

High-throughput component placement

The key to our optimized placement is the ability to project system throughput under each placement strategy, using a three-step process.

First, by knowing the component profile and input workload characteristics, we can learn the mapping between the input request rate $l_{workload}$ and component runtime resource demands $(r_{cpu}, f_{cpu}, r_{network}, f_{network}, f_{mem})$.

Second, given a component placement strategy, we can derive the maximum input request rate that can saturate the CPU, network bandwidth, or memory resources at each server. For CPU and network bandwidth, we can use either the average or peak resource needs to derive the rate. Let $t_{CPU\ average}$, $t_{CPU\ peak}$, $t_{network\ average}$, $t_{network\ peak}$, and t_{memory} denote such saturation rates at a server. Components collocated on the same server must share the host CPU and memory resources, while the intraserver component communication can exploit high-bandwidth IPC (interprocess communication) mechanisms.

Third, we can estimate the system throughput as the lowest saturation rate for all resource types at all servers. Using the average resource needs, we can derive an optimistic throughput estimate:

$$\min_{\text{for all servers}} \left\{ \tau_{\text{CPU average}}, \tau_{\text{network average}}, \tau_{\text{memory}} \right\}$$

Using the peak resource needs, we can derive a pessimistic throughput estimate:

$$\min_{\text{for all servers}} \left\{ \tau_{\text{CPU peak}}, \tau_{\text{network peak}}, \tau_{\text{memory}} \right\}.$$

Because we can project system throughput under any placement strategy, we can discover a high-throughput component placement through exhaustive search. However, the search space for all possible placement strategies can be very large for applications with many components over many servers. When exhaustive search becomes computationally infeasible, low-complexity optimization algorithms become necessary. Previous studies have proposed such algorithms for certain constrained cases.[4,5]

Business logic constraints might impact the placement policy. In this case, we simply remove invalid placement candidates from the search space, and select the high-throughput choice from the remaining candidates.

Runtime component migration

To estimate runtime component resource needs, we need knowledge of runtime dynamic workload characteristics. Our system will feed this information into the *component placement executive* (see Figure 1) to assist dynamic placement decisions. Some component middleware systems can be instrumented to trace intercomponent messages.[6,7] To provide less intrusive monitoring, we could employ network-level packet sniffing.

Marcos Aguilera and his colleagues propose performance debugging for multicomponent online applications based on message traces.[8] They analyze intercomponent causal message paths and then derive the corresponding component response time. However, they use precollected message traces, and they don't explicitly address the acquisition of runtime workload characteristics. Odysseytrades application resource demands for service quality, or *fidelity*, using a history-based prediction of resource demands.[9] In comparison, because we use component profiles, we need only to monitor the input workload characteristics at runtime, which is cheaper and more accurate than directly predicting runtime resource demands.

Besides performance optimization, runtime component migration must consider two issues.

First, to achieve high scalability and fault tolerance, the component placement executive can have a decentralized or even peer-to-peer architecture. For instance, servers can share components' runtime workload characteristics and profiles with their peers. Participating servers can then independently make component migration decisions that they consider beneficial.

Second, system stability is important, especially when migration decisions are decentralized. To achieve a certain level of stability, we can employ a component migration threshold such that the system performs only those migrations that produce an over-the-threshold benefit. A careful balance must be maintained between responsiveness and system stability.

Preliminary results

To test the effectiveness of profile-driven component placement, weperformed experiments based on the RUBiS (Rice University Bidding System) benchmark.[10] RUBiS is an auction site prototype modeled after eBay; it implements such a site's core functionality: selling, browsing, and bidding. It follows the three-tier Web service model, containing a front-end Web server, movable business logic components, and a back-end database. We used the Enterprise JavaBeans (EJB) version of RUBiS with bean-managed persistence. We provide results here only for static placement; we'll investigate dynamic component migration in the future.

We conducted profiling and experiments on a Linux cluster connected by a 100-Mbytes-per-second Ethernet switch. Each server had dual 1.26-GHz Pentium III processors and 2 Gbytes of memory. A JBoss 3.2.3 application server with an embedded Tomcat 5.0 servlet container hosted the RUBiS EJB components. The database server ran MySQL 4.0. The data set was approximately 1 Gbyte; we sized it according to database dumps published on the RUBiS Web site, http://rubis.objectweb.org.

Component profiling

RUBiS contains 11 components: the Web server, the database, and nine EJB components implementing the auction service logic (`Bid, BuyNow, Category, Comment, Item, Query, Region, User,` and `UserTransaction`). During our profiling runs, each component ran on a dedicated server, and we measured the component resource consumption at request rates ranging from one to 11 requests per second. Our request mix was 15 percent read-write requests and 85 percent read-only requests, which is similar to the mix that Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel use.[10] We used a modified SYSSTAT toolkit in the measurement. For the peak CPU and network usage, we recorded the 90-percentile values for the measured rates at one-second intervals.

Once we had the resource consumption measurements, we derived general functional mappings using linear fitting. Figures 2 and 3 show such a derivation for the `Bid` component and the Web server. Table 1 lists the profiling results for all 11 components. We don't show the memory-profiling results because we can't precisely measure the component memory consumption using SYSSTAT . This didn't affect the component placement decisions for this experiment because the server memory wasn't the bottleneck resource under any placement.

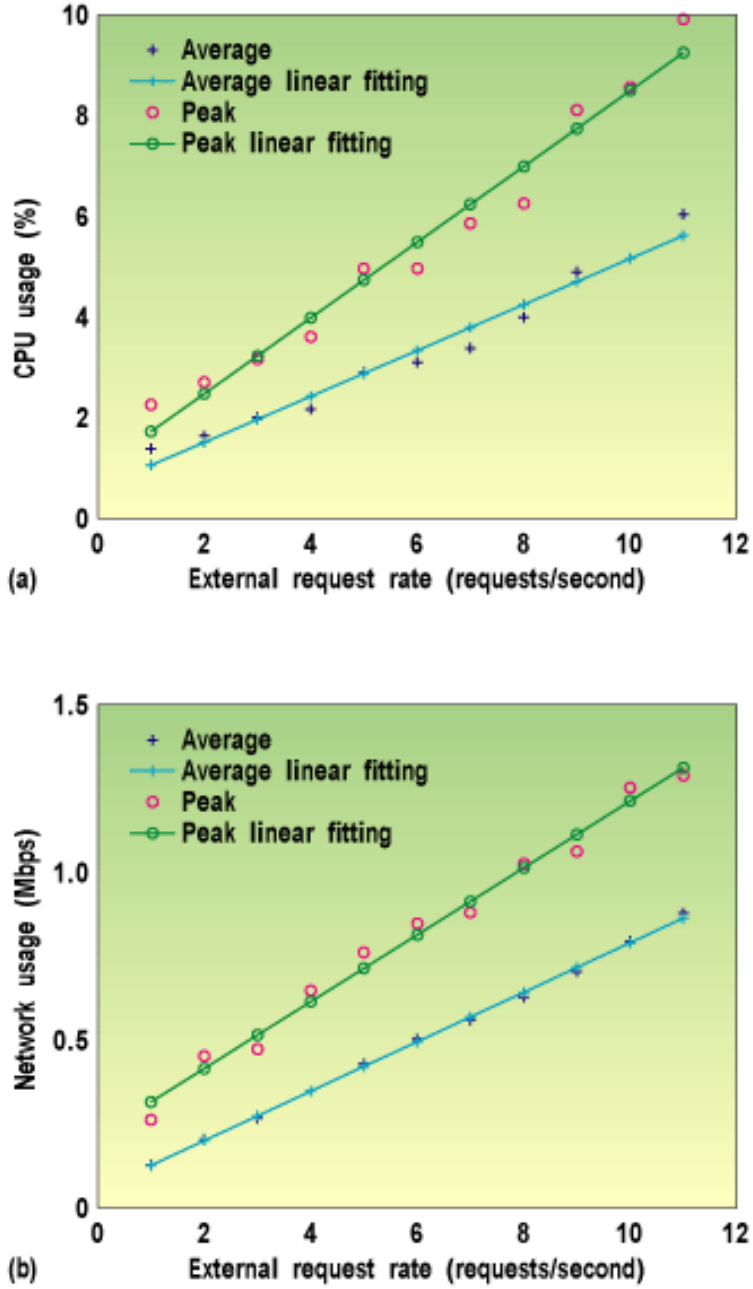

Figure 2. Linear fitting for the `Bid` component: (a) CPU average and peak usage; (b) network average and peak usage.

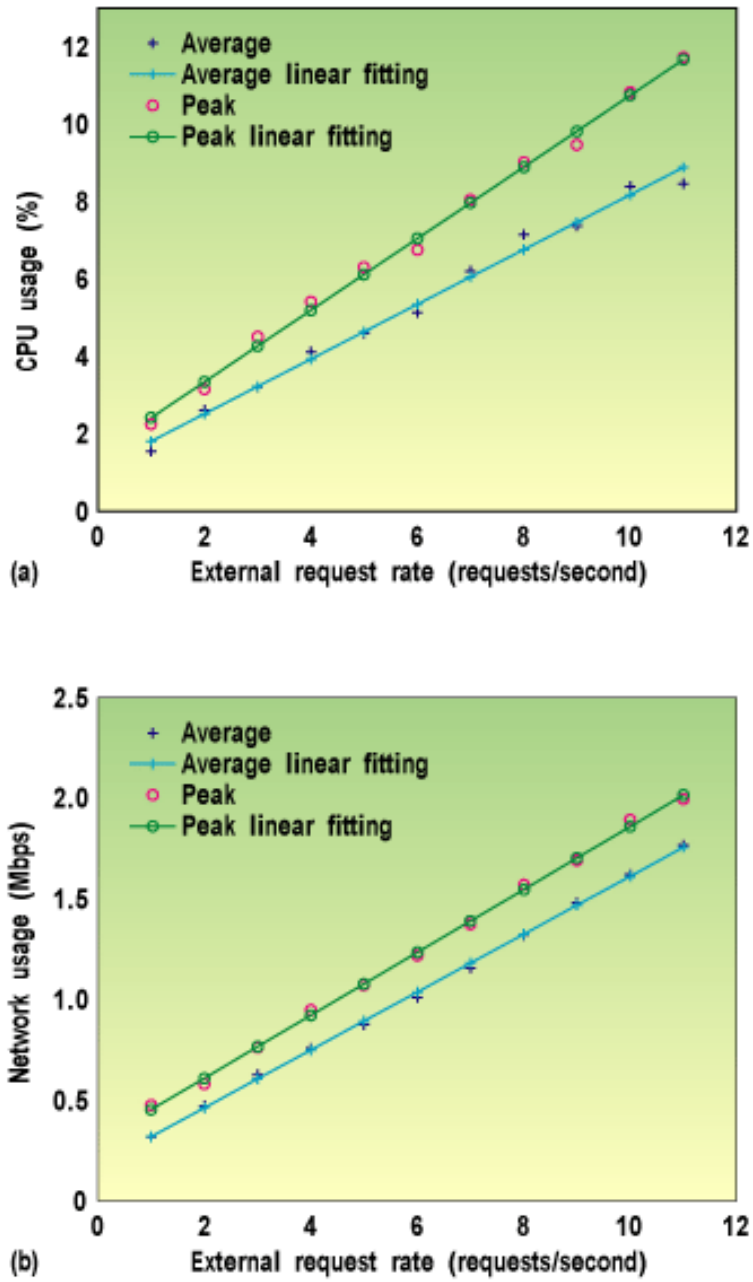

Figure 3. Linear fitting for the Web server: (a) CPU average and peak usage; (b) network average and peak usage.

Table 1. Component profiles for RUBiS (Rice University Bidding System), based on linear fitting of measured resource usage at 11 input request rates. $\lambda_{workload}$ is the average request arrival rate (in requests/second).

	CPU usage (%)		Network usage (Mbps)	
Component	**Average**	**Peak**	**Average**	**Peak**
Web server	$0.707 \cdot \lambda_{workload} + 1.101$	$0.925 \cdot \lambda_{workload} + 1.488$	$0.144 \cdot \lambda_{workload} + 0.175$	$0.156 \cdot \lambda_{workload} + 0.297$
Database	$0.012 \cdot \lambda_{workload} + 0.875$	$0.061 \cdot \lambda_{workload} + 0.778$	$0.008 \cdot \lambda_{workload} + 0.063$	$0.009 \cdot \lambda_{workload} + 0.218$
Bid	$0.456 \cdot \lambda_{workload} + 0.594$	$0.752 \cdot \lambda_{workload} + 0.967$	$0.074 \cdot \lambda_{workload} + 0.053$	$0.100 \cdot \lambda_{workload} + 0.216$
BuyNow	$0.006 \cdot \lambda_{workload} + 0.494$	$0.033 \cdot \lambda_{workload} + 0.775$	$0.000 \cdot \lambda_{workload} + 0.049$	$0.000 \cdot \lambda_{workload} + 0.213$
Category	$0.000 \cdot \lambda_{workload} + 0.912$	$0.000 \cdot \lambda_{workload} + 1.093$	$0.000 \cdot \lambda_{workload} + 0.072$	$0.000 \cdot \lambda_{workload} + 0.217$
Comment	$0.004 \cdot \lambda_{workload} + 0.817$	$0.000 \cdot \lambda_{workload} + 0.898$	$0.000 \cdot \lambda_{workload} + 0.049$	$0.000 \cdot \lambda_{workload} + 0.214$
Item	$0.306 \cdot \lambda_{workload} + 1.605$	$0.528 \cdot \lambda_{workload} + 1.949$	$0.074 \cdot \lambda_{workload} + 0.151$	$0.088 \cdot \lambda_{workload} + 0.259$
Query	$0.000 \cdot \lambda_{workload} + 0.898$	$0.000 \cdot \lambda_{workload} + 0.987$	$0.000 \cdot \lambda_{workload} + 0.055$	$0.000 \cdot \lambda_{workload} + 0.214$
Region	$0.104 \cdot \lambda_{workload} + 1.023$	$0.266 \cdot \lambda_{workload} + 1.065$	$0.041 \cdot \lambda_{workload} + 0.053$	$0.052 \cdot \lambda_{workload} + 0.219$
User	$0.091 \cdot \lambda_{workload} + 1.040$	$0.221 \cdot \lambda_{workload} + 1.130$	$0.036 \cdot \lambda_{workload} + 0.045$	$0.046 \cdot \lambda_{workload} + 0.206$
UserTransaction	$0.000 \cdot \lambda_{workload} + 0.803$	$0.000 \cdot \lambda_{workload} + 0.903$	$0.000 \cdot \lambda_{workload} + 0.048$	$0.000 \cdot \lambda_{workload} + 0.213$

High-throughput component placement

We examined the component placement problem on two servers. We placed the restriction that the Web server and the database are never collocated on one server. Because we could place each of the remaining nine components on either the Web server or the database server, this setup had 512 possible component configurations. We then projected the optimistic and pessimistic system throughput for each of the 512 configurations. We chose the placement with the best pessimistic performance (called *profiler's choice*). In this placement, `Query`, `Region`, and `User` are collocated with the Web server while the other EJB components are collocated with the database.

We compared this placement with three other strategies based on common heuristics. The first two strategies placed all EJB components with the Web server or with the database, respectively. The third strategy (called *writers with Web*) placed all read-write components (that is, `Bid, BuyNow, Comment, Item,` and `UserTransaction`) with the Web server and the read-only components with the database. The intuition behind

this strategy was that read-only components tend to interact more with the database.

Figure 4 shows the RUBiS throughput under different levels of input workload. In our experiments, we counted a request as successful only if it returned within eight seconds. We define a placement's throughput as the highest throughput achieved at any input request rate. The results show that profiler's choice outperformed all other placement strategies by over 30 percent. Performance degraded slightly after the system attained maximum throughput. This is because some requests exceeded the time-out limit of eight seconds despite being partially completed.

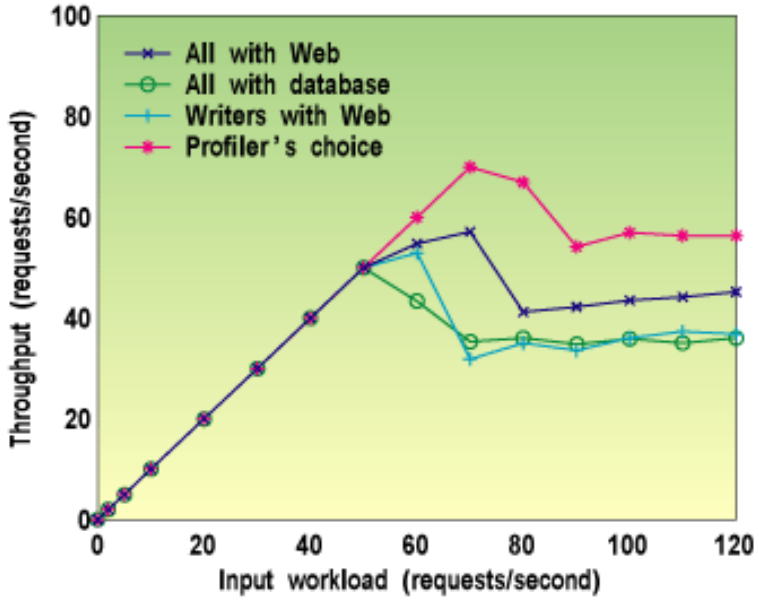

Figure 4. The performance of different component placement strategies for RUBiS (Rice University Bidding System).

Figure 5 illustrates the accuracy of pessimistic and optimistic throughput estimations for the four placement strategies. On average, the pessimistic estimation is 39 percent smaller than the optimistic estimation. The measurement results almost always fall between the two estimations. Component profiles constructed using more fine-grain resource usage measurements (for example, LTT) should improve the accuracy of the throughput estimations. Taking other factors into consideration, such as component context switch and remote-invocation overhead, might also improve accuracy. We plan to investigate these issues.

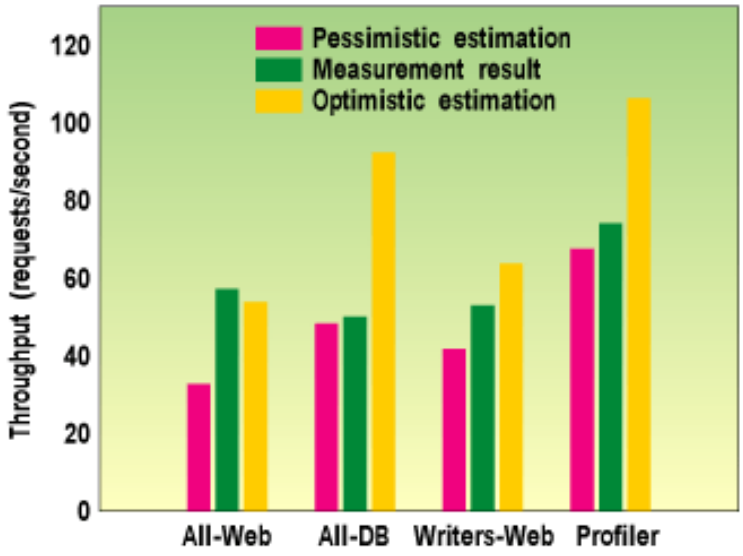

Figure 5. The accuracy of throughput estimations.

Conclusion

Future work will proceed on two principal fronts. First, we'll study heuristic placement for heterogeneous systems with many components and nodes. As we mentioned before, we'll also investigate runtime workload monitoring and dynamic online component migration to handle changing input workload characteristics.

Acknowledgments

We thank the members of the University of Rochester Computer Science systems group and the anonymous referees for their valuable comments. This work was supported partly by NSF grants EIA-0080124, CCR-0204344, CCR-0219848, ECS-0225413, CCR-0306473, and ITR/IIS-0312925; by the US Dept. of Energy Office of Inertial Confinement Fusion under Cooperative Agreement DE-FC03-92SF19460; and by equipment or financial grants from IBM and Sun Microsystems Laboratories.

References

1. A. Fox , et al., "Cluster-Based Scalable Network Services,"*Proc. 16th ACM Symp. Operating Systems Principles* (SOSP 97), ACM Press, 1997, pp. 78-91.

2. K. Shen , et al., "Integrated Resource Management for Cluster-Based Internet Services,"*Proc. 5th USENIX Symp. Operating Systems Design and Implementation* (OSDI 2002), USENIX, 2002,pp. 225-238; www.usenix.org/publications/library/proceedings/osdi02/tech/shen.html.

3. K. Yaghmour and M.R. Dagenais, , "Measuring and Characterizing System Behavior Using Kernel-Level Event Logging,"*Proc. 2000 USENIX Ann. Tech. Conf.,* USENIX, 2000; www.usenix.org/publications/library/proceedings/usenix2000/general/full_papers/yaghmour/yaghmour.pdf.

4. G.C. Hunt and M.L. Scott, , "The Coign Automatic Distributed Partitioning System,"*Proc. 3rd USENIX Symp. Operating Systems Design and Implementation* (OSDI 1999), USENIX, 1999; www.usenix.org/publications/library/proceedings/osdi99/full_papers/hunt/hunt.pdf.

5. R.K. Balan , et al., "Tactics-Based Remote Execution for Mobile Computing,"*Proc. 1st USENIX Int'l Conf. Mobile Systems, Applications, and Services* (MobiSys 03), USENIX, 2003; www.usenix.org/publications/library/proceedings/mobisys03/tech/full_papers/balan/balan.pdf.

6. M.Y. Chen , et al., "Pinpoint: Problem Determination in Large, Dynamic Internet Services,"*Proc. Int'l Conf. Dependable Systems and Networks* (DSN 02), IEEE CS Press, 2002,pp. 595-604.

7. R.J. Stets , G.C. Hunt, and M.L. Scott , "Component-Based APIs for Versioning and Distributed Applications,"*Computer,*July 1999,pp. 54-61.

8. M.K. Aguilera , et al., "Performance Debugging for Distributed Systems of Black Boxes,"*Proc. 19th ACM Symp. Operating Systems Principles* (SOSP 03), ACM Press, 2003,pp. 74-89.

9. D. Narayanan and M. Satyanarayanan, , "Predictive Resource Management for Wearable Computing,"*Proc. 1st USENIX Int'l Conf. Mobile Systems, Applications, and Services* (MobiSys 2003), USENIX, 2003; www.usenix.org/publications/library/proceedings/mobisys03/tech/full_papers/narayanan/narayanan.pdf.

10. E. Cecchet , J. Marguerite, and W. Zwaenepoel, , "Performance and Scalability of EJB Applications,"*Proc. 17th ACM Conf. Object Oriented Programming Systems, Languages, and Applications* (OOPSLA 02), ACM Press, 2002,pp. 246-261.

Christopher Stewart is enrolled in the PhD program at the University of Rochester. He investigates problems in large distributed systems, commodity operating systems, and power-aware computing. He received his BS in computer science from Morehouse College. Contact him at the Computer Science Dept., Univ. of Rochester, Rochester, NY 14627-0226; stewart@cs.rochester.edu.

Kai Shen is an assistant professor in the University of Rochester's Department of Computer Science. He received his PhD in computer science from the University of California at Santa Barbara. His research interests are in parallel and distributed systems, operating systems, and computer networks. He's a member of the ACM and USENIX. Contact him at the Computer Science Dept., Univ. of Rochester, Rochester, NY 14627-0226; kshen@cs.rochester.edu.

Sandhya Dwarkadas is an associate professor of computer science and of electrical and computer engineering at the University of Rochester. Her research interests lie in parallel and distributed systems, computer architecture, and the interaction of compilers, architectures, and run-time systems. She received her PhD in electrical and computer engineering from Rice University. Contact her at the Computer Science Dept., Univ. of Rochester, Rochester, NY 14627-0226; sandhya@cs.rochester.edu.

Michael L. Scott is a professor at and past chair of the University of Rochester's Department of Computer Science. His research interests span operating systems, languages, architecture, and tools, with a particular emphasis on parallel and distributed systems. He received his PhD in computer science from the University of Wisconsin-Madison. He's a member of the ACM and a senior member of the IEEE. Contact him at the Computer Science Dept., Univ. of Rochester, Rochester, NY 14627-0226; scott@cs.rochester.edu.

Jian Yin is a research staff member at the IBM T.J. Watson Research Center. His research interests include operating systems, distributed systems, and fault tolerance. He received his PhD from the University of Texas at Austin. Contact him at the IBM T.J. Watson Research Center, Hawthorne, NY 10532; jianyin@us.ibm.com.

Research on Distributed-Component Placement

Our work is related to a number of prior studies on distributed-component placement. Coign examines the optimization problem of minimizing communication time for two-machine client-server applications.[1] ABACUS focuses on the placement of I/O-specific functions for cluster-based data-intensive applications.[2] Addistant[3] and J-Orchestra[4] support partitioning and distributing execution of "legacy" Java applications through byte code rewriting. DVM (Distributed Virtual Machine) further adds security to such support.[5] Except for Coign, these projects focus on mechanisms for transparent remote execution, leaving placement decisions largely to users.

Anca-Andreea Ivan and her colleagues examine the automatic deployment of component-based software over the Internet, subjected to throughput requirements.[6] In their approach, application developers must specify each component's resource requirements. Bhuvan Urgaonkar and his colleagues study the benefit of letting applications overbook CPU and network resources in shared hosting platforms.[7] Their work is limited to the placement of multiple single-component server applications that don't interact.

The Aura[8] and Chroma[9] projects propose exporting system-level mobility information to user-level agents, which can then make strategic decisions (including component placement) based on their model of user activity and intent. User-level involvement in placement systems seems particularly valuable for "intelligent" applications, in which user intent is central and closely tied to mobility. For online services, however, we emphasize transparent system-level management.

References

1. G.C. Hunt and M.L. Scott, , "The Coign Automatic Distributed Partitioning System,"*Proc. 3rd USENIX Symp. Operating Systems Design and Implementation* (OSDI 1999), USENIX, 1999; www.usenix.org/publications/library/proceedings/osdi99/full_papers/hunt/hunt.pdf.

2. K. Amiri , et al., "Dynamic Function Placement for Data-Intensive Cluster Computing,"*Proc. 2000 USENIX Ann. Tech. Conf.,* USENIX, 2000; www.usenix.org/events/usenix2000/general/full_papers/amiri/amiri.pdf.

3. M. Tatsubori , et al., "A Bytecode Translator for Distributed Execution of 'Legacy' Java Software,"*Proc. 15th European Conf. Object- Oriented Programming* (ECOOP 2001), LNCS 2072, Springer-Verlag, 2001, pp. 236–255.

4. E. Tilevich and Y. Smaragdakis, , "J-Orchestra: Automatic Java Application Partitioning,"*Proc. 13th European Conf. Object-Oriented Programming* (ECOOP 2002), LNCS 2374, Springer-Verlag, 2002, pp. 178–204.

5. E.G. Sirer , et al., "Design and Implementation of a Distributed Virtual Machine for Networked Computers,"*Proc. 17th ACM Symp. Operating Systems Principles* (SOSP 99), ACM Press, 1999,pp. 202–216.

6. A.-A. Ivan , et al., "Partitionable Services: A Framework for Seamlessly Adapting Distributed Applications to Heterogeneous Environments,"*Proc. 11th IEEE Symp. High Performance Distributed Computing* (HPDC 02), IEEE CS Press, 2002,pp. 103–112.

7. B. Urgaonkar , P. Shenoy, and T. Roscoe, , "Resource Overbooking and Application Profiling in Shared Hosting Platforms,"*Proc. 5th USENIX Symp. Operating Systems Design and Implementation* (OSDI 2002), USENIX, 2002,pp. 239–254; www.usenix.org/publications/library/proceedings/osdi02/tech/urgaonkar.html.

8. J.P. Sousa and D. Garlan, , "Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments,"*Proc. 3rd Working IEEE/IFIP Conf. Software Architecture* (WICSA 2002), IEEE CS Press, 2002, pp. 29–43.

9. R.K. Balan , et al., "Tactics-Based Remote Execution for Mobile Computing,"*Proc. 1st USENIX Int'l Conf. Mobile Systems, Applications, and Services* (MobiSys 03), USENIX, 2003; www.usenix.org/publications/library/proceedings/mobisys03/tech/full_papers/balan/balan.pdf.

Research on Application-Resource-Consumption Profiling

Bhuvan Urgaonkar and his colleagues use application-resource-usage profiling to guide application placement in shared hosting platforms.[1] Cristiana Amza and her colleagues provide bottleneck resource analysis for three dynamic online-service benchmarks.[2] Xiaohui Gu and Klara Nahrstedt examine quality-of-service-aware multimedia service partitioning and placement based on service dependency graphs.[3] Although their application-profiling research is somewhat similar to our research, our component profiles provide a more detailed characterization of the mapping between input request rate and application resource consumption. Such information is critical to making high-throughput placement decisions when resource needs are workload dependent.

A recent study by Ron Doyle and his colleagues models the service response time reduction with increased memory cache size for Web servers.[4] However, such modeling is only feasible with intimate knowledge about application-memory-usage behavior. It doesn't share our goal of maintaining general applicability on a wide range of applications.

References

1. B. Urgaonkar , P. Shenoy, and T. Roscoe, , "Resource Overbooking and Application Profiling in Shared Hosting Platforms,"*Proc. 5th USENIX Symp. Operating SystemsDesign and Implementation* (OSDI 2002), USENIX, 2002,pp. 239–254; www.usenix.org/publications/library/proceedings/osdi02/tech/urgaonkar.html.

2. C. Amza , et al., "Specification and Implementation of Dynamic Web Site Benchmarks," *Proc. 5th IEEE Workshop Workload Characterization* (WWWC 5), 2002.

3. X. Gu and K. Nahrstedt, , "Dynamic QoS-Aware Multimedia Service Configuration in Ubiquitous Computing Environments," *Proc. 22nd Int'l Conf. DistributedComputing Systems* (ICDCS 2002), IEEE CS Press, 2002,pp. 311–318.

4. R.P. Doyle , et al., "Model-Based Resource Provisioning in a Web Service Utility,"*Proc. 4th USENIX Symp. Internet Technologies and Systems* (USITS 03), USENIX, 2003; www.usenix.org/publications/library/proceedings/usits03/tech/full_papers/doyle/doyle.pdf.

Cite this article: Christopher Stewart, Kai Shen, Sandhya Dwarkadas, Michael L. Scott, and Jian Yin, " Profile-Driven Component Placement for Cluster-Based Online Services," IEEE Distributed Systems Online, vol. 5, no. 10, 2004.