MassConf: Automatic Configuration Tuning By Leveraging User Community Information*

Wei Zheng, Ricardo Bianchini, Thu D. Nguyen
Department of Computer Science, Rutgers University
110 Frelinghuysen Road, Piscataway, NJ 08854, USA
{wzheng, ricardob, tdnguyen}@cs.rutgers.edu

ABSTRACT

Configuring modern enterprise software can be extremely difficult because their behaviors often depend on large numbers of configuration parameters. Software vendors can simplify the configuration process for new users by collecting and using configuration information from existing users. In particular, we observe that (1) a "good" configuration may work well for many different users, and (2) multiple configurations may work well for each user. We leverage these observations to design MassConf, a system that collects and uses existing configurations to automatically configure new software installations. Our evaluations with a case study confirm our observations and show that MassConf successfully reaches the targets of many more new installations than an existing efficient optimization algorithm.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*

General Terms

Algorithms, Design, Experimentation, Management, Performance

Keywords

Automatic configuration

1. INTRODUCTION

Enterprise software is becoming increasingly complex. A single piece of server software may include hundreds of configuration parameters, as software vendors and contributors (collectively called "vendors" hereafter) want their systems to be as flexible and adaptable as possible. Selecting proper values for configuration parameters is critical, since they may affect the software's behavioral correctness, performance, availability, and/or energy consumption.

Unfortunately, configuring modern software can be extremely difficult since a good configuration depends (at least) on the hardware environment, the workload, the load intensity, and the target behavior (e.g., some level of performance or availability). Moreover, there can be relationships between the parameters that are not made explicit by the software documentation. Thus, it is very hard (if not impossible) for users to completely understand the configuration-hardware-workload-intensity-target relationships.

Due to the size and complexity of this configuration space, previous research has focused on approaches and tools to detect and troubleshoot misconfigurations [1, 10, 11, 15, 19, 20, 21], to study the resilience of systems in the face of configuration errors [9], to automatically configure machines in a single installation [2, 3], and to automatically tune configurations for best performance [18, 22].

Although these efforts have been useful, a user's ability to configure her software to achieve a certain target behavior is still far from ideal in practice. For example, a user who wants her server software to produce an average response time of 50 milliseconds is left clueless, when the default configuration reaches only 100 milliseconds. As long as the parameters that affect performance are identified, this user can run existing algorithms (e.g., [12]) to optimize the server's performance by experimentation with different configuration settings. However, tuning performance may involve a very large number of time-consuming experiments [18, 22].

Vendors need to do more to help users configure their software. One possible approach is to create automatic configurers that run locally at the users' sites and select the best values for the parameters, either through experimentation or modeling. A simpler and cheaper approach, the one we advocate here, is for the vendor to collect configuration information from the existing user community of its software and use it in configuring the software for new users.

Our proposed approach is based on two key observations. First, *a configuration may actually work well for many users*, i.e., it may work well for many workloads, load intensities, and target behaviors, especially when the users use similar hardware platforms. The second observation is that *multiple configurations may actually work well for each user*, i.e., they may all meet the user's target behavior. Together, these observations mean that there is flexibility in which existing configuration to select for each new user.

Further, any work that users may do to tune their configurations can benefit new users of the software. Thus, we

*This research was partially supported by NSF grant CNS-0916878.

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

propose to leverage the existing users' configurations to find a good configuration for each new user. To demonstrate this idea, we designed MassConf, a system that automatically collects configuration and environment information from existing users, clusters users according to environment, produces an ordered (ranked) list of possible configurations, and tests each configuration in turn at the new user's site until the target behavior is met. After the configuration of each new user is complete, MassConf may change the ranking of configurations. MassConf seeks to (1) reach the target behavior for as many new users as possible and (2) minimize the average number of experiments required at the new users' sites.

The most interesting technical aspect of MassConf is its ranking of configurations. Faced with our first observation above, one might be tempted to rank configurations based on popularity; more popular configurations would be tried first at the new users' sites. The popularity information is readily available from the existing users' deployed configurations. However, as our experiments shall demonstrate, the popularity-based ranking is not the best choice. The reason is that particularly effective but difficult-to-find configurations would tend to appear towards the end of the list. Ranking them higher would allow more new users to be configured with fewer experiments.

To account for this effect, MassConf would require information about how every deployed configuration would do for every existing user. Unfortunately, this information is obviously not obtainable. Thus, MassConf adapts its behavior over time, by moving the configuration that is selected for each new user toward the front of the ranked list, regardless of its actual popularity. This adaptation increases the chance that another new user will also experiment with (and hopefully benefit from) the selected configuration.

An optimized version of MassConf, called MassConf+, improves ranking further by pruning the ranked list of configurations after an initial "learning" period. Shortening the list rids MassConf+ of configurations that are unlikely to satisfy a large number of new users, thereby reducing the average number of experiments. We do not discuss MassConf+ further in this paper because of space constraints; details about MassConf+ can be found in [23].

To evaluate MassConf's ranking of configurations in an interesting (yet understandable) case study, we investigate its use for automatically configuring the Apache Web server to achieve a response-time target. As a baseline for comparison, we use Simplex, an efficient algorithm [12] that has been successfully used to optimize server software [4, 22].

Our evaluation results show that adaptive ranking requires many fewer experiments than popularity-based ranking to configure a population of new users. In fact, we find that the faster we move a selected configuration to the front of the ranking, the better on average. We also find that MassConf can configure many more new users than Simplex. Moreover, MassConf requires many fewer experiments than Simplex, even when we consider only those new users that both systems can configure.

Our findings illustrate that software configuration can be significantly simplified by having users contribute parts of their configurations and use them to configure the software for other users. Because of its simplicity and effectiveness, we conclude that MassConf and its adaptive configuration ranking have great potential to work well in practice.

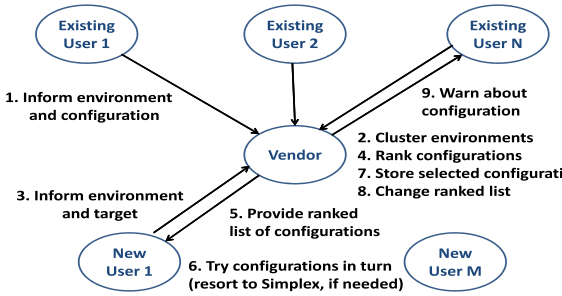

Figure 1: MassConf overview.

2. MASSCONF

2.1 Design

Figure 1 illustrates the MassConf design. The next few paragraphs detail each part of the design in turn.

Data collection from existing users. MassConf is run by the software vendor. First (step 1 in the figure), it collects configuration and environment information from each existing user that is willing to participate. (Although some users may refuse to participate, many would likely be willing to contribute since they can benefit from it as shall be clear below.) This information is extracted by instrumentation in the server software itself and sent to the vendor.

The configuration information describes the setting of each configuration parameter of the software. The settings can be of any type, e.g., boolean, numeric, or character strings. When the configuration information may include sensitive data, only a few relevant parameter values may be collected. (The vendor should know which parameters may include sensitive data.)

As part of the configuration information, MassConf must be informed about the users' *high-level goals* when they selected their configurations. For example, the goal may have been to improve performance, improve performability (performance + availability), or lower energy consumption.

MassConf stores the parameter settings it receives without modification, except in the case of numeric parameters. For each numeric parameter, MassConf breaks the range of possible values into 10 evenly sized chunks. Two configurations are grouped together if their values for each parameter fall in the same chunk. For example, suppose that each configuration has two parameters, p_1 and p_2, with possible values ranging from 1 to 200 (chunks of size 20) and from 1 to 100 (chunks of size 10), respectively. Further, suppose that the values of these parameters for configurations C_4 and C_5 are: $C_4(p_1) = 10$ (first chunk), $C_4(p_2) = 18$ (second chunk), $C_5(p_1) = 16$ (first chunk), and $C_5(p_2) = 12$ (second chunk). Because the chunks match for all parameters, C_4 and C_5 would be grouped together.

The configurations in each group are represented by a single "average" configuration. In the average configuration, each parameter is given the average of the values seen for that parameter in the corresponding group. For example, the average configuration C_{avg} for the cluster formed by configurations C_4 and C_5 above would have $C_{avg}(p_1) = 13$ and $C_{avg}(p_2) = 15$.

The environment information is a description of the hardware (e.g., number of cores, amount of memory) and possibly the low-level software (e.g., operating system, settings for relevant environment variables) at the user's site. This

information is necessary since the behavior of the software to be configured may depend heavily on the environment.

Clustering existing users according to environment. Using the environment information, MassConf then clusters the existing users (step 2) as was done in Mirage [5] for software upgrade deployment. The idea is to cluster users that have similar environments together, so that their configuration information can be used for new users with similar environments. For example, the vendor of a multithreaded server may want to separate out user sites with vastly different numbers of cores or threading libraries, as these aspects of the environment may have a significant effect on the ideal number of threads with which to configure the server. Conversely, user sites with similar numbers of cores and thread libraries should be clustered together. A number of algorithms can be used for clustering, but we prefer the Quality Threshold (QT) algorithm [8]. QT starts with one site per cluster. It then iteratively adds sites to clusters (effectively merging clusters) while trying to achieve the smallest average inter-site distance and not to exceed a pre-defined maximum cluster diameter. The algorithm stops when no more clusters can be merged together. Our distance metric involves the aspects of the environment that differ between clusters. Each aspect is weighted by the vendor, according to its importance to the software configuration.

Collecting information from a new user. When configuring a new user, MassConf first deploys the software to the new user's site and collects its environment information (step 3). Then, it requests from the user a description of the software's target behavior (step 3). The target behavior reveals the high-level goal for the configuration tuning. With the new user's environment information, MassConf can now identify the best cluster for it.

Ranking configurations. Using the configuration information from this cluster, MassConf produces a ranked list (or ranking) of configurations to be tried at the new user's site (step 4). The list is formed by the configurations of the existing users that had the same goal as the new user. (For example, we do not want to use information about configurations that were selected to lower energy consumption when configuring servers for maximum throughput.) The exact ordering of the list is influenced by the order of the users' (both new and existing) arrivals, as described below. The list is transferred to the new user's site (step 5). At this point, MassConf can run experiments with each configuration, until the desired behavior is met or it runs out of configurations to try (step 6).

Testing configurations at the new user's site. These experiments are run under the user's actual workload and load intensity, so the user herself may have to provide a realistic test harness to exercise the software. If all experiments are run and the desired behavior is never achieved, the user is warned. MassConf's inability to reach a target may mean that the target is unrealistic for the workload and load intensity, or that it still does not have enough information (i.e., enough existing users) to produce a large enough coverage of the possible configurations. If the user confirms that the target is achievable and the parameter values are numeric, MassConf resorts to Simplex, *starting from the best configuration it has found so far*. However, we expect that MassConf would rarely have to resort to other approaches in practice; in most cases, the new user would relax the target. In these cases, MassConf would most likely have already found an appropriate configuration.

Storing the selected configuration. When a configuration is selected, MassConf includes information about it in its central database of existing users (step 7). At that point, the new user becomes one of the existing users within the corresponding cluster. (As MassConf found the configuration for the new user, it already has all the information required from an existing user.) Thus, after the bootstrapping period, the population of existing users should exhibit similar characteristics (as a group) to the new users (also taken as a group).

Adapting the ranking. As it is impossible to predict the set of new users that will want to join the system, MassConf adjusts its ranking (step 8) by moving the configurations that have been selected for each new user towards the top of the ranking. This adjustment enables very good configurations to be chosen more often. When MassConf needs to resort to Simplex, the new configuration is added to the end of the ranking. We discuss these decisions in detail below.

Providing feedback to existing users. Finally, MassConf warns existing users when their configurations seem suboptimal (i.e., new users with the same goal have selected other configurations) with respect to the rest of the users in the same cluster (step 9). This feedback to existing users is an incentive for them to provide their configuration and environment information, even when they had to configure their software entirely by hand or when the community was still small. Another important incentive may be to help these users configure an upgraded version of the software by leveraging information from the users who benefited most from MassConf to configure the existing version.

2.2 Configuration Ranking

Dynamically adapting the ranking. As mentioned above, a popularity-based ranking can be misleading. It is possible that unpopular configurations can actually satisfy many more new users than popular ones. The reason these highly useful configurations are not more popular may be that they are harder to find, e.g., they are only needed for heavy workloads or hard-to-achieve target behaviors.

Thus, MassConf dynamically adapts its rankings to eventually concentrate configurations that can satisfy many users at the top. We study three approaches for promoting the selected configurations within a ranking: *slow*, *fast*, and *fastest*. The slow approach moves a selected configuration one slot up in the ranking. The fast approach moves the configuration to the halfway point between its current slot and the top of the ranking. The fastest approach moves the configuration directly to the first slot of the ranking. Figure 2 shows an example of how ranking (a) is adjusted after configuration C_4 and C_5 are selected by two consecutive new users, using the slow (b), fast (c), and fastest (d) adaptation approaches. For example, in the fast approach, C_4 is first moved from the 8th to the 4th slot in the ranking. This moves C_5, C_9, C_1, and C_8 one slot down the ranking. Then, when C_5 is selected by the next new user, it moves from the 5th to the 3rd slot. This moves C_3 and C_4 one slot down.

Regardless of the speed of promotion, any new configurations that are added to the system are appended to the end of the corresponding ranking. The reason is that we want

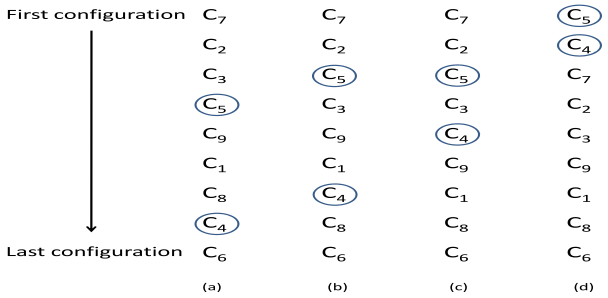

Figure 2: Original ranking (a) and slow (b), fast (c), and fastest (d) adaptation approaches, after configurations C_4 and C_5 are selected by two consecutive new users.

to see more than one user benefit from a new configuration before we promote it up the ranking.

Note that configurations from existing users are treated the same as those selected for new users, despite the fact that the former users select their configurations by means other than MassConf. We also considered the possibility of not altering the ranking when an existing user joins with a configuration that had already been seen. We ultimately decided against this approach because it would disregard the fact that the configuration satisfied an additional user.

2.3 Bootstrapping

MassConf may only start to perform well when the existing users within each cluster become a good representation of the new users to come into the same cluster. Until that point, MassConf may be unable to meet the target behavior requested by new users without resorting to Simplex. A new user may also decide to optimize the configuration manually until the target behavior is achieved. Fortunately, these Simplex-derived or manually generated configurations contribute to MassConf just the same as the configurations of existing users that join MassConf.

3. CASE STUDY: CONFIGURING APACHE

We consider the configuration of the Apache Web server to achieve a target average response time as a case study to understand and evaluate MassConf. We study a synthetic population of users representing a pessimistic scenario for MassConf because we lack real configuration data.

3.1 Methodology

We briefly present our evaluation methodology, referring the interested reader to [23] for more details.

Apache configuration parameters. We configure the five main parameters that affect Apache performance: StartServers, MinSpareServers, MaxSpareServers, MaxClients, and MaxRequestsPerChild [17].

Workloads, intensities, and targets. Each synthetic user represents a different combination of workload, load intensity, and response-time target. Each workload is defined by its fraction of requests for three types of content, small static files, a cached large static file, and dynamic CGI scripts, designed to stress the file system, networking, and CPU, respectively. Then, for each workload, we assign load intensities from 50 requests/sec (rps) to an experimentally determined maximum throughput (using the default configuration) with a step of 50 rps. Also, for each workload, we select different targets 5% apart within the range bounded by the performance achieved using the default configuration and the best performance achievable using Simplex.

User populations. We generate 219 "existing users" evenly spread in the 3D space of workloads, load intensities, and response-time targets. We define the existing users' configurations by running Simplex to find configurations that meet the users' response-time targets.

We generate 195 "new users" who are also evenly spread across the parameter space, but are completely distinct from the existing users. Our goals are to select configurations for as many of these new users as possible, while using the smallest possible number of experiments on average.

Our evenly spread and non-overlapping populations of users represent a pessimistic scenario for MassConf. The reason is that any concentration of users in specific parts of the workload-intensity-target space would increase the likelihood that (1) many users would deploy the same configuration, and (2) many users could be satisfied by each configuration, both of which would improve MassConf's performance.

Platform. Our experiments are run on two PCs, each running Linux 2.6.18 and containing a 2.8 GHz Xeon CPU, 2 GB of memory, and a 7200 rpm disk. One machine hosts an HTTP client emulator and the other the Apache Web server (version v2.0.4). In each experiment, the client sends a pre-defined load intensity with Poisson request arrivals.

3.2 Understanding Configuration Ranking

Figure 3 plots the popularity of the configurations (of the 5 parameters listed above) that met the performance targets for the existing users. A configuration has popularity of x% if it is used by x% of the users. The X-axis shows the index of the unique configurations in decreasing order of popularity while the Y-axis shows the cumulative popularity. The leftmost point is the default configuration.

The figure confirms one of the basic premises of MassConf, namely that certain configurations work well for many existing users. Specifically, the default configuration works well for a large fraction of users. In addition, the fact that the curves are not straight lines shows that other configurations are also used by multiple users.

As we have suggested, however, ranking configurations based on popularity only may miss very good configurations that just happen to be unpopular. Figure 4 illustrates this effect clearly. The X-axis lists the index of each unique existing configuration in decreasing order of popularity (from left to right). Only the default configuration (index #0) is not listed. The Y-axis lists the number of new users in our population that could be satisfied by each configuration.

The default configuration can satisfy the performance targets of 66 new users. As the figure shows, there is another configuration (#50) that can satisfy even more new users (70) but is very unpopular. This means that 49 other configurations would be tried before reaching this very good one when using a popularity-based ranking. A similar observation can be made of configuration #20, which can satisfy 58 new users. In contrast, the two most popular configurations can only satisfy 17 and 11 new users, respectively. These observations provide clear motivations for MassConf's adaptive ranking approach.

3.3 Experimental Evaluation

We now turn to evaluating the use of MassConf for configuring new user installations. We initialize MassConf with

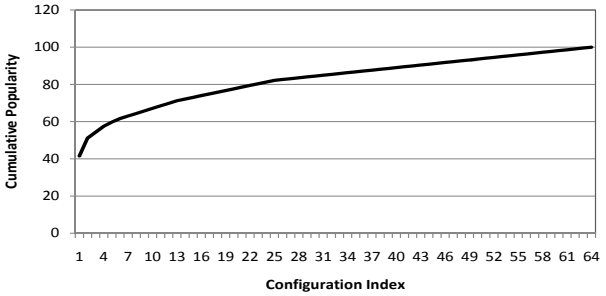

Figure 3: Popularity of configurations among existing users.

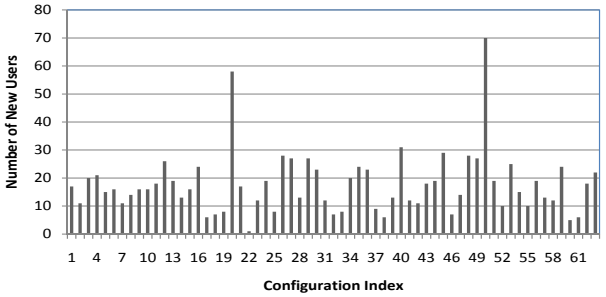

Figure 4: Popularity ranking and number of new users that can be satisfied by each configuration.

Table 1: MassConf vs. popularity and optimal static for the 129 new users requiring different configurations than the default.

# of Exp's	Popularity Ranking	MassConf Adapt-fast	MassConf Adapt-fastest	Optimal Static
Total	1519	1380	1272	873
Avg.	11.8	10.7	9.9	6.8
Max.	84	84	84	84

configurations from our 219 existing users and use it to configure our 195 new users to meet their response-time targets.

We compare MassConf's adaptive ranking algorithms to popularity-based ranking. (Note that the ranking also changes dynamically in popularity-based ranking whenever the selection of an existing configuration causes the popularity ordering to change.) We also compare MassConf's performance against Simplex (running on its own starting from the default configuration). In our experiments, we set Simplex to terminate when the target average response time is reached or the standard deviation of the vertices' response times is smaller than 5 milliseconds [12].

In addition, we present results for the "optimal" static ranking, i.e., the static ranking that generates the smallest possible number of experiments in configuring the new users. This ranking sorts the configurations in decreasing order of the number of new users that they satisfy. Obviously, the optimal ranking can only be determined because we know the entire set of new users in advance, which is impossible in practice. We present results for the optimal ranking simply as a lower bound on the number of experiments.

Table 1 summarizes some of the results. Since the behaviors of MassConf's adaptation algorithms depend on the exact sequence in which new users join the system, we generated 10 random sequences and averaged the results. We make several observations from our experiments.

1. MassConf successfully reached the performance targets of all new users. Out of our 195 new users, 66 were able to meet their response-time targets using the default configuration. MassConf was able to configure *all* 129 new users that could not use the default configuration.

2 and 3. Adaptive ranking beats popularity-based ranking. The faster the adaptive algorithm promotes configurations, the better. Table 1 shows that MassConf with Adapt-fast and Adapt-fastest ranking algorithms require fewer experiments on average than the popularity-based ranking. The analysis of adaptive ranking from the previous section suggested this result. In fact, the faster selected configurations are promoted up the ranking, the smaller the average number of experiments per user. Our complete results [23] show that *the best adaptive ranking (Adapt-fastest) runs up to 24% fewer experiments per user than popularity-based ranking on average.* In contrast, Adapt-slow (not shown in Table 1) requires up to 85% more experiments per user than popularity-based ranking on average. There are two effects at play here: (1) on the positive side, moving a good configuration up enables it to satisfy more users; and (2) on the negative side, it may increase the number of experiments required when a configuration that was moved down is selected. When moving up one slot at a time, only a few extra users can be satisfied by the promoted configuration, so the negative effect becomes more prominent. When moving configurations up faster, the good configurations can satisfy many extra users, making the positive effect more prominent.

The fact that Adapt-fastest is the best approach confirms the two observations that motivated our MassConf design: some configurations work well for multiple users and multiple configurations work well for each user. If only one configuration met each user's target, Adapt-fastest would make the worst decision. At the other extreme, if all configurations met all new users' targets, all approaches would produce the same number of experiments, i.e., 1.

MassConf Adapt-fastest is 31% slower compared to the (unrealistic) optimal static ranking. However, MassConf+ actually outperforms this static ranking because it shortens the list of configurations to be tried [23].

4. MassConf successfully reached the performance targets for many more users than Simplex. As mentioned above, MassConf was able to configure all 129 new users that could not use the default configuration. In contrast, *Simplex failed to configure 74 of these new users.* Even when MassConf is not allowed to resort to Simplex, it still can configure 67 more new users than Simplex (122 vs. 55). Simplex cannot configure many new users because it gets stuck at local minima, trying configurations that lead to very similar performance.

The ability of MassConf to configure many more new users than Simplex is particularly interesting since our existing user configurations were originally derived using Simplex. This result reinforces the point that Simplex has to search a large space of configurations each time it is used and so, for any particular search, it may miss some "good" configurations. MassConf is completely different in that it is guided by the tuning efforts of existing users and its adaptive ranking algorithms.

5. MassConf is faster than Simplex. We compare the number of experiments required by MassConf and Simplex for the subset of 55 new users for which both approaches

were able to achieve the performance targets. Averaging over 10 random sequences of new users, MassConf with the three adaptive ranking approaches require between 13.3 and 17.8 experiments per user on average. The best approach, Adapt-fastest, requires 13.3 experiments on average, which is 28% faster than Simplex (18.6).

4. RELATED WORK

Leveraging existing data on configurations. Several previous works have investigated how to leverage others' configurations to diagnose and troubleshoot misconfigurations [1, 16, 19, 20].

Even though MassConf also relies on configuration information from a population of users, it focuses on a completely different problem: configuration tuning; there are no misconfigurations to troubleshoot. As detailed in Section 2, the impact of this key difference is that our main focus has been on issues that have not been addressed before, namely the study of adaptive ranking algorithms and the average number of experiments to which they lead.

Configuration tuning. Many works have considered the performance tuning of server configurations, e.g. [4, 6, 7, 18, 22]. Osogami *et al.* [13, 14] focused on shortening each experiment, rather than reducing the number of experiments.

MassConf differs from these works in four main ways: (1) it seeks to produce configurations that meet the users' target behaviors, rather than to find the best possible configuration; (2) it relies on configuration information from a population of systems, rather than a single system; (3) it relies on adaptive ranking algorithms to tune performance efficiently; and (4) unless it needs to resort to Simplex, it tests existing configurations for new users, rather than trying to use experience or dependencies to create new configurations.

5. CONCLUSIONS

In this paper, we addressed the problem of configuring enterprise software efficiently. Specifically, we proposed MassConf, a system that uses existing configurations to automatically configure the software for new users. The configuration process relies on dynamic adaptation of the order of configurations (ranking) to be tried. To evaluate MassConf, we used it to configure Apache to achieve the performance targets of a population of users. The results showed that our fastest adaptation leads to the smallest number of experiments. The results also showed that MassConf is able to configure more users in fewer experiments than Simplex, an efficient optimization algorithm.

Acknowledgements. We would like to thank Kai Shen and Christopher Stewart for their comments on this work.

6. REFERENCES

[1] B. Aggarwal et al. NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge. In *Proceedings of NSDI '09*, April 2009.

[2] P. Anderson et al. SmartFrog meets LCFG: Autonomous Reconfiguration with Central Policy Control. In *Proceedings of LISA '03*, Oct. 2003.

[3] M. Burgess. Cfengine: A Site Configuration Engine. *USENIX Computing systems*, 8(3), 1995.

[4] I. Chung et al. Automated Cluster-Based Web Service Performance Tuning. In *Proceedings of HPDC '04*, June 2004.

[5] O. Crameri et al. Staged Deployment in Mirage, an Integrated Software Upgrade Testing and Distribution System. In *Proceedings of SOSP '07*, October 2007.

[6] Y. Diao et al. Managing Web Server Performance with AutoTune Agent. *IBM Systems Journal*, 42(1), 2003.

[7] S. Duan et al. Tuning Database Configuration Parameters with iTuned. In *Proceedings of VLDB '09*, August 2009.

[8] L. J. Heyer et al. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, 1999.

[9] L. Keller et al. ConfErr: A Tool for Assessing Resilience to Human Configuration Errors. In *Proceedings of DSN '08*, June 2008.

[10] E. Kiciman et al. Discovering Correctness Constraints for Self-Management of System Configuration. In *Proceedings ICAC '04*, May 2004.

[11] K. Nagaraja et al. Understanding and Dealing with Operator Mistakes in Internet Services. In *Proceedings of OSDI '04*, December 2004.

[12] J. A. Nelder et al. A Simplex Method for Function Minimization. *Computer Journal*, 7(4), 1965.

[13] T. Osogami et al. Finding Probably Better System Configurations Quickly. In *Proceedings of SIGMETRICS '06*, June 2006.

[14] T. Osogami et al. Optimizing System Configurations Quickly by Guessing at the Performance. *SIGMETRICS Perform. Eval. Rev.*, June 2007.

[15] C. Stewart et al. Performance Modeling and System Management for Multi-component Online Services. In *Proceedings of NSDI '05*, May 2005.

[16] Y. Su et al. AutoBash: Improving Configuration Management with Operating System Causality Analysis. In *Proceedings of SOSP '07*, October 2007.

[17] The Apache Software Foundation. Apache HTTP Server Version 2.0. http://httpd.apache.org/docs/2.0/mod/prefork.html.

[18] R. Thonangi et al. Finding Good Configurations in High-Dimensional Spaces: Doing More with Less. In *Proceedings of MASCOTS '08*, September 2008.

[19] H. J. Wang et al. Automatic Misconfiguration Troubleshooting with PeerPressure. In *Proceedings of OSDI '04*, Dec. 2004.

[20] Y. Wang et al. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *Proceedings of LISA '03*, Oct. 2003.

[21] A. Whitaker et al. Configuration Debugging as Search: Finding the Needle in the Haystack. In *Proceedings of OSDI '04*, Dec. 2004.

[22] W. Zheng et al. Automatic Configuration of Internet Services. In *Proceedings of Eurosys '07*, March 2007.

[23] W. Zheng et al. MassConf: Automatic Configuration Tuning By Leveraging User Community Information. Technical Report DCS-TR-664, Department of Computer Science, Rutgers University, January 2010.