Benchmarking and Configuration of Workflow Management Systems

Michael Gillmann, Ralf Mindermann, Gerhard Weikum
University of the Saarland, Germany
e-mail: {gillmann,minderma,weikum}@cs.uni-sb.de
WWW: http://www-dbs.cs.uni-sb.de

Abstract. Workflow management systems (WFMS) are a cornerstone of mission-criticial, possibly cross-organizational business processes. For large-scale applications both their performance and availability are crucial factors, and the system needs to be properly configured to meet the application demands. Despite ample work on scalable system architectures for workflow management, the literature has neglected the important issues of how to systematically measure the performance of a given system configuration and how to determine viable configurations without resorting to expensive trial-and-error or guesswork. This paper proposes a synthetic benchmark for workflow management systems; based on the TPC-C order-entry benchmark, a complete e-commerce workflow is specified in a system-independent form. This workflow benchmark, which stresses all major components of a workflow system and is parameterized in a flexible manner, has been applied to two operational systems, the commercial system Staffware97 and our own prototype system Mentor-lite. The paper reports performance results from our measurements and discusses lessons learned. In particular, the results demonstrate the scalability of the Mentor-lite architecture. The measurements also underline the need for configuring systems intelligently, and the paper outlines an auto-configuration tool that we have been building to this end.

1   Introduction

1.1   Problem Statement

Workflow technology has penetrated into mission-critical, enterprise-wide or even cross-organizational, business applications. Typical examples are insurance claim processing, cargo shipping, or healt-care tracking and planning, and workflow technology is also embedded in many e-commerce services. Following the terminology of WfMC [32] (see also [5, 6, 8, 15, 18]), a workflow is a set of activities that belong together in order to achieve a certain business goal. Activities can be completely automated or based on interaction with a human user and intellectual decision-making. In particular, an activity can spawn requests to an arbitrary "invoked application" that is provided by some server independently of the current workflow. Workflow management systems (WFMS) orchestrate the control and data flow between a workflow's activities, based on a high-level specification of the intended behavior (e.g., using Petri-net variants, state charts, or some script language) with some leeway for exception handling and run-time improvisation (as needed, e.g., in medical applications).

Despite their business success, most WFMS products exhibit specific idiosyncracies and, by and large, significant deficiencies and limitations in terms of their performance. The current situation is probably comparable to that of relational database systems in the eighties. Also and similarly to database technology, configuring and tuning a WFMS for satisfactory performance falls more in the realm of black art (i.e., guesswork or expensive trial-and-error

experimentation) and sorely lacks scientific foundations. Even such mundane basics such as systematic benchmarks are still missing.

1.2   Contributions

This paper aims to improve the state of the art on the systematic assessment and judicious configuration of WFMSs by defining a simple yet powerful benchmark workflow, based on an e-commerce scenario. The benchmark specification takes the well-known TPC-C order-entry application as a starting point, combines the three major transactions of TPC-C into a workflow, and extends this setting to systematically stress all components of a WMFS architecture. The benchmark is parameterized for adaptation to different scales and application scopes. It is specified using the statechart formalism [13], and can be easily converted into the specification languages of most WFMSs.

To demonstrate the platform-independence and usefulness of the benchmark, we present performance measurements of two systems, the commercial product Staffware97 and our own prototype system Mentor-lite. These experiments have revealed limitations and system-specific bottlenecks with regard to scalability, thus underlining the need for such benchmarking. An additional, important lesson is that proper system configuration is a key issue, and that performance depends quite sensitively on subtle aspects of the configuration.

Out of the benchmarking efforts, we have started work towards an "intelligent" auto-configuration tool for WFMS architectures, with specific emphasis on the Mentor-lite architecture. In our prior work [9] we have focused on the analytical underpinnings for such a tool; in the current paper we present the architecture of the tool itself and its interaction with the various components of the WFMS environment. The implementation of the tool is close to completion and we plan on evaluating the quality of its recommendations using our e-commerce benchmark.

1.3   Related Work

Although the literature includes much work on scalable WFMS architectures [1, 5, 6, 8, 15, 19], there are only few research projects that have looked into the quantitative assessment of WFMS configurations. The work reported in [2, 3] presents several types of distributed WFMS architectures and discusses the influence of different load distribution methods on the network and workflow-server load, mostly using simulations. [28] presents heuristics for the allocation of workflow-type and workflow-instance data onto servers. Mechanisms for enhanced WFMS availability by replicating state data on a standby backup server have been studied in [11, 17]. [16] characterizes the workload of cross-organizational workflows by means of Markov models. None of this prior work has addressed the issue of how to systematically benchmark a WFMS and how to configure a WFMS for given performance goals.

Benchmarking is well established, with benchmark specifications gradually evolving, as a form of systematic performance assessment for processors and compilers [14], and also in the area of database and transaction processing systems [10, 25]. In particular, the suite of benchmarks published by the Transaction Performance Council (TPC) [30] has proven extremely useful for driving performance enhancements of commercial database systems over the last decade. On the other hand, the work of the TPC also shows that defining a systematic benchmark involves many subtle pitfalls and presents a quite challenge. In particular, finding a careful balance between making the benchmark realistic and functionally comprehensive and ensuring that it can be installed and run on many different platforms with affordable effort is all but trivial.

For the WFMS area and even for modern middleware in general, benchmarking is in its infancy. [20] presents a benchmark for a specific system, namely, Lotus Notes, but this effort solely focuses on Notes's use as a message/mail engine. [4] is even more specialized in its performance study of scientific lab processes in genome research. SAP has its product-specific benchmark suite [26] that stresses also the R/3-internal workflow engine, but this benchmark is

completely tailored to the SAP environment, involves many issues of an ERP system that are not related to workflow technology, and would be difficult to port to another WFMS. The very recent TPC-W benchmark [31] considers Web-based e-commerce, but emphasizes the routing, multiplexing, load balancing, caching, and security capabilities of Web application servers (of the category such as Apache, IIS, etc.) and pays no attention to the workflow aspects of e-services. Finally, [7] has made a laudable effort to define a general benchmark for active database systems, but has indeed restricted itself to the core functions of an active rule engine and cannot be generalized to the richer spectrum of WFMS services.

1.4   Paper Outline

The rest of the paper is organized as follows. Section 2 discusses the general rationale for a systematic benchmark of advanced WFMS architectures. Section 3 presents the specification of the e-commerce benchmark. Section 4 briefly reviews the architecture of the two systems under test. Section 5 presents the setup of the experimental testbed. Section 6 shows the results of our measurements. Section 7 discusses major lessons learned. Section 8 presents the architecture of the auto-configuration tool that we are building.

2   Benchmark Rationale

2.1   Metrics for Performance Assessment

From a business perspective, the benefit of a WFMS should ideally be measured in terms of how smooth and cost-effective the system runs a company's business processes. This involves issues like how much the WFMS contributes to customer satisfaction, effectivity of office workers (e.g., call-center agents), meeting deadlines, reducing operational cost, and ultimately the company's success in the market. Obviously, despite some recent business research along these lines (e.g., [27]), such benefits are hard to quantify. Our goal is much more limited in scope, focusing on metrics that can be directly attributed to the computer-support infrastructure of business processes.

Similar to transaction-processing benchmarks, a key metric is obviously the **throughput** of the entire WFMS. Unlike OLTP, our notion of throughput has a much larger granularity and extension in time: we measure throughput in terms of completed workflows (i.e., instances of a given workflow type), and we have to collect these numbers separately for different workflow types. Note that it does not make sense to amalgate throughput figures for a complex workflow type that involves week-long activities (e.g., processing a credit request in a bank) and a simple, short workflow type that usually completes within a few minutes (e.g., opening a bank account) into a single, seemingly unified metric, which could, however, no longer be meaningfully interpreted.

When we know the number of clients that interact with the WFMS, the rates at which these initiate new workflows, and the relative frequencies of the different workflow types, then the throughput that the system can sustain must be at least as high as the aggregate load induced from all clients. In addition, it is equally crucial that the WFMS is sufficiently responsive. In the context of long-lived business processes, this entails the following two subgoals. (1) The **turnaround time** for an entire workflow must be acceptable. Obviously this depends also on the efficiency of the human users that process intellectual activities, but it is important that the WFMS does not incur any additional bottlenecks. Also, and importantly for benchmarking, the turnaround time from the initiation of a workflow to its completion, as perceived by the initiating user, is the key metric for fully automated workflows that contain no interactive activities. (2) For workflows that include interactive activities and could thus have an extended lifetime of several days or weeks, the critical units are the individual interactions between a user and the WFMS that occur during activities and, especially, at the start and end of an activity when role resolution and other worklist-handling tasks are performed. For the purpose of our benchmark definition we are thus interested in the **step response time**, where a step could be a specific type of user interaction or an entire activity.

2.2   Components under Test

A WFMS is a complex system that consists of a build-time component for specifying workflows, a run-time component with the actual **workflow engine** for the proper interpretation of control and data flow in between activities, and a suite of administration tools for monitoring etc. For our purpose, the run-time engine is the component that matters. In most WFMSs, the workflow engine is run as a multithreaded server process, or sometimes as a collection of processes on the same or different computers. This workflow server may have its own persistent storage system for tracking the states of long-lived workflows, or more typically relies on a **database server** for this purpose. In addition, it may interact with **application servers** for invoked applications that result from automated activities. Finally, the communication among these various servers may be based on other, lower-level, middleware that is itself implemented in the form of dedicated **communication servers**. The latter serve as a reliable request brokers, with TP monitors (MQ Series, MTS, etc.) or ORBs (e.g., Orbix) being prevalent examples.

With the workflow engine really being the heart of a WFMS, it is crucial for a benchmark to stress the entire functional spectrum of the engine. Therefore, one requirement is to exercise the different types of **control-flow constructs**, notably, conditional branching, fork-join parallelism, and also loops. These should be supported by all industrial-strength WFMSs, but it is widely open to what extent they are handled efficiently.

An equally important, orthogonal, aspect to be examined by the benchmark is the support for different types of activities. Fully **automated activities**, which may invoke an **external application** on an application server, and **interactive activities** that run on client machines pose fairly different requirements on the workflow engine, and some commercial WFMS are known to be particularly geared for one type of activities.

Our benchmark will include all major types of control-flow constructs and both types of activites. However, the benchmark will be designed such that different, restricted, levels of specialization can be derived when the interest is on specific application requirements or special-purpose WFMSs (e.g., for call-center applications or canonical, simple types of e-commerce). The components under test will include the workflow server as well as any underlying database or storage servers and also external application servers and communication servers if these are present in a given WFMS architecture. The actual external applications (e.g., the program for testing the authenticity of a hand-written or digital signature) will, however, only be emulated by a stub (i.e., a "dummy" program) inside the application server. Similar to the functional specializations, we will allow different scoping levels of the benchmark to either include or exclude such additional components. So, in the simplest case the benchmark solely examines the workflow engine, and in the most advanced case it takes the full suite of surrounding software and the corresponding servers into account.

3   Benchmark Specification

In this section we describe our benchmark specification. The benchmark reflects all previously described metrics and tests all interesting system components. Our proposal captures an e-commerce scenario. It is similar to the TPC-C benchmark for transaction systems [30], with the key difference that we combine multiple transaction types into a workflow and further enhance the functionality by control and data flow handling. Furthermore, we explicitly take application invocations into account.

The control flow specification is given in the form of state charts [12, 13]. This specification formalism has been adopted for the behavioral dimension of the UML industry standard, and it has been used for our own prototype system Mentor-lite [22, 33].

Figure 1 shows the top-level state chart for our e-commerce (EC) benchmark workflow. Each state corresponds to an activity or one (or multiple, parallel) subworkflow(s). We assume that for every activity *act* the condition *act*_DONE is set to true when *act* is finished. So, we are able to synchronize the control flow so that a state of the state chart is left when the corresponding activity terminates. For parallel subworkflows, the final states of the

corresponding orthogonal components serve to synchronize the termination (i.e., join in the control flow).
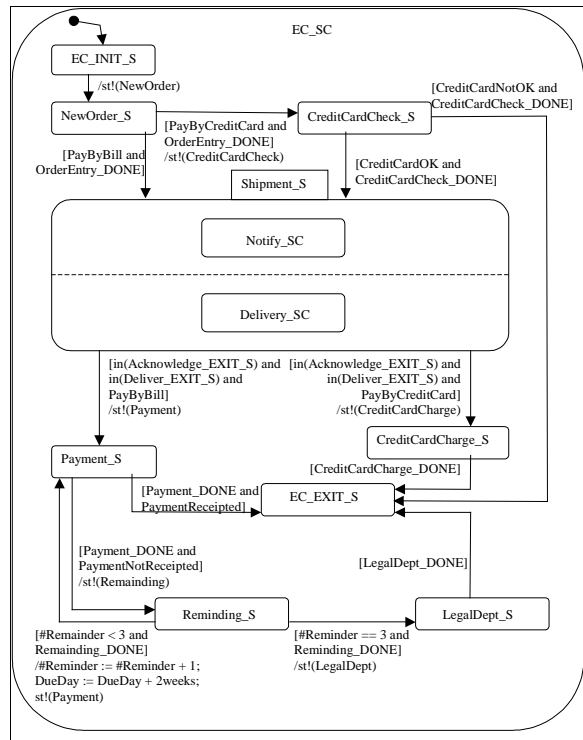

Figure 1 : State chart of the *electronic commerce (EC)* workflow example

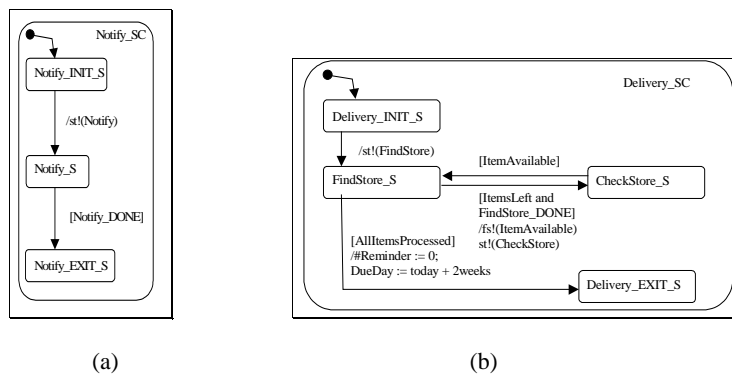(a)   (b)

Figure 2 : State charts of the *Notify* and the *Delivery* subworkflows

The workflow behaves as follows. After the initialization of the workflow instance, the *NewOrder* activity is started. After the termination of *NewOrder*, the control flow is split. If the customer wants to pay by credit card, the condition *PayByCreditCard* is set and the *CreditCardCheck* activity checks the validity of the credit card. If there are any problems with the credit card, the workflow is terminated. Otherwise the shipment, represented by the nested

top-level state *Shipment_S*, is initiated spawning two orthogonal/parallel subworkflows, specified in the state charts *Notify_SC* (Figure 2a) and *Delivery_SC* (Figure 2b), respectively. The first subworkflow *Notify* has only one activity that sends an acknowledgment mail. The delivery of goods is done by one or several external, eventual autonomous stores. So, the second subworkflow, *Delivery*, (sequentially) invokes for each ordered item an activity that identifies a store from which the item could be shipped. Then, a second activity instructs the store to deliver the item and waits for an acknowledgement. The both activities *FindStore* and *CheckStore* are repeated within a loop over all ordered items. If the customer wants to pay by bill, a reminder counter and the due day for the payment have to be initialized. After the termination of both subworkflows, the control flow is synchronized, and split again depending on the mode of payment. If the customer wants to pay by credit card, the credit card is now charged and the workflow terminates. If the customer wants to pay by bill, an activity is invoked that waits for the settlement of the invoice. If the payment is confirmed within the running period, the workflow also terminates. If the payment is not confirmed after two weeks, within a loop an activity is invoked that sends a reminder to the customer. Moreover, the reminder counter is increased and the due day initialized again. This is repeated at most three times. If the payment is not receipted within the period after the third reminder, an activity is invoked that inform the legal department and the workflow terminates. Note, that we have so far neglected the exception handling (e.g. if an item is not deliverable).

The EC workflow specification is straightforward to implement when the WFMS that is to be benchmarked uses state charts. In this case only the stubs for the simulated application programs have to be implemented, but this can be done with one generic stub with different parameter settings. Installing the benchmark on other WFMSs is still simple when the workflow specification language of the target WFMS is a well-founded "high-level" formalism such as Petri nets, event-process chains, etc. In fact, automatic conversion of state charts into other formalisms are largely feasible [22, 24].

4 Systems under Test

4.1 Mentor-lite

The core of our research prototype Mentor-lite [21] is an *interpreter* for state chart specifications. The interpreter performs a stepwise execution of the workflow specification according to its formal semantics [33]. For each step, the activities to be performed by the step are determined and started. Two additional components, the *communication manager* and the *log manager*, are closely integrated with the workflow interpreter. All three components together form the *workflow engine*. The execution of a workflow instance can be distributed over several workflow engines at different sites. A separate *workflow log* is used at each site where a Mentor-lite workflow engine is running. The communication manager is responsible for sending and receiving synchronization messages between the engines. These messages contain information about locally raised events, updates of state chart variables and state information of the local engine [23]. When a synchronization message is received, the corresponding updates at the receiving site are performed. In order to guarantee a consistent global state even in the presence of site or network failures, Mentor-lite uses the CORBA Object Transaction Services (OTS) to implement reliable message queues. The CORBA implementation Orbix provides the basic communication infrastructure for distributed execution. The workflow engine, comprising the three components interpreter, communication manager, and log manager, is implemented as an Orbix server. Its IDL interface provides a method to start a workflow instance and a method to set variables and conditions within the workflow instance.

Databases like the *workflow repository* (i.e., a repository of workflow specifications) or the *worklist database* can be shared by Mentor-lite workflow engines at different sites. In the current setup, the underlying DBMS is Oracle 7.

Communication interfaces to application programs are implemented by wrappers using the distributed computing environment CORBA. On top of these interfaces, protocols for complex interactions with application programs are specified in terms of state and activity charts. The workflow engine starts the wrappers asynchronously and uses the methods of the wrapper objects to read or set variables. The application wrappers can in turn use the workflow engine's method to set control flow variables.

In this paper, we consider two different versions of the Mentor-lite implementation. The difference between the two releases is in the handling of application invocations. The first version, referred to as "*ml-proc*", starts a new process for each external application on the workflow-server site for the data exchange with the external application. The advantage of this approach is the increased availability of the system as only one workflow is affected when the communication with the application fails. However, the required main memory on the workflow server site increases significantly. The second version of Mentor-lite, referred to as "*ml-thr*", uses threads within a single process, which is much more memory-efficient.

4.2   Staffware97

Staffware97 has a client-server architecture with a monolithic server. All components like log manager, worklist handler, etc. are implemented inside the *workflow engine*. The workflow engine can be run on multiple workflow servers, but each workflow instance is bound to one server and exclusively handled by this server throughout its lifetime. So Staffware97 does not support a partitioned and distributed workflow execution. The workflow engine manages several *work queues* that start application programs and can be polled by the user's clients.

A work queue schedules the work items of one or several users in a FIFO manner with priorities, i.e., FIFO among all processes with the same priority. The number of users per work queue as well as the number of parallel processes per user are system parameters. This is a critical issue especially for mostly automated workflows, i.e., workflows with mostly automated, non-interactive activities, because all such activities are scheduled in the work queue of a single user (i.e., dummy user "auto").

Staffware97 provides an interface to application programs based on Dynamic Data Exchange (DDE). External application programs are called via scripts that use DDE commands. Automated, non-interactive activities can also be started without the use of any scripts but only under the restriction that they run on the workflow engine's server machine.

The exchange of data between the workflow engine and the application programs is handled via the file system. Input data to the application as well as results from the application are stored in and read from temporary files. In an asynchronous application invocation, the calling script has to raise an *event* when the workflow engine is allowed to read the result data. Automated, non-interactive programs running on the server machine are able to communicate directly with the workflow engine also via *pipes*. In our measurements, we used both options for comparison. We refer to the file- and event-based as "*sw-ev*", and to the pipe-based alternative as "*sw-pi*".

Note that Staffware97 is no longer the current release of Staffware's workflow product, but the newer version Staffware2000 [29] became available only very recently.

5   Experimental Testbed

For every WFMS under test we must create its full-fledged system environment with all necessary servers, including middleware components (e.g. Corba or a TP-monitor), since even for simulated external applications the benchmark includes the invocation and completion steps. So the testbed is significantly more complex than for TPC-C-like benchmarks.

Our testbed consists of the following five modules. (1) A synthetic load generator starts new workflow instances with a given interarrival time distribution. In our case, we used a Poison arrival process with a given mean as a parameter. (2) A monitoring component observes and logs the start and stop times of the activities and entire workflows. (3) Stub applications

simulate the external applications. These stubs simply read their input data, sleep for a specified time period, and return control-flow-relevant result data. The mean turnaround time of these stubs is a parameter of the experiments. (4) A dedicated SUN Sparc5 is used as an application server connected to the workflow-server machine with an Ethernet LAN. The use of a separate machine for the application server, as opposed to running applications on the workflow-server machine, was optional and varied in the experiments. (5) The workflow server itself runs on a dedicated SUN Sparc10 with all, WFMS-specific, additional components as described in following. In all experiments reported here, we limited ourselves to a single workflow server (i.e., did not make use of multiple workflow engines or even partitioned and distributed workflow execution).

Mentor-lite additionally needed the Oracle7 database system for its logging and recovery component and as a repository containing workflow specifications etc. The database was not dedicated for the WFMS and ran on a remote database server machine. For the communication interfaces of the workflow engine and the application wrappers we used Orbix 2.3.

Staffware97 offers a choice between logging in a database or to the file system. We chose the file system variant. In the experiments that involved a separate application server, the application invocation was done by a remote-shell call from a script on the workflow-engine machine.

In our baseline experiment, the system configuration consists of exactly one server machine. So, all components of the WFMS including the external application programs were run on the dedicated SUN Sparc10. The turnaround time of the activities was normally distributed with a mean of 10 seconds and a standard deviation of 4 seconds.

In the second series of measurements, we studied the impact of the system configuration. Specifically, we added a dedicated computer for the application server and the external applications.

As mentioned before, our e-commerce benchmark supports different test levels with regard to functionality and the scope of the benchmarking. For example, we can easily derive specialized, simplified variants that contain only automated activities or no loops. In the measurements presented here, we specialized the benchmark by using only the control flow path for the case of credit card payment, disregarding the reminder loops, and limiting the delivery loop to exactly one iteration. As a consequence, the workflow contained only automated activities, no interactive ones.

6   Experimental Results

As mentioned in Section 4, we benchmarked two versions of Mentor-lite, *ml-proc* and *ml-thr*, and two versions of Staffware97, *sw-ev* and *sw-pi*. In all measurements, we used an observation window of 8 hours.

The baseline experiment used one dedicated computer for all servers. The mean turnaround time of the activities was set to 10 seconds each.
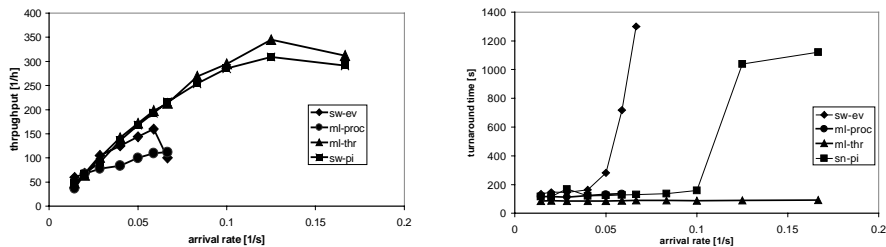

Figure 3 : Throughput and turnaround time of baseline experiment

Figure 3 shows the system throughput in completed workflow instances per hour as a function of the workflow arrival rate, and the mean turnaround time of workflow instances in seconds. As expected, the throughput increases linearly with the arrival rate until the system saturates. The maximum sustainable throughput of *ml-thr* is about 10% higher than that of *sw-pi*. The other two candidates under test performed poorly. *sw-ev* exhibited very high delays because of waiting for events to read result data from the file system. The low throughput of *ml-proc* resulted from hardware and operating-system bottlenecks, especially with regard to memory, for too many active processes (one per activity) had to be handled.

The turnaround time of Mentor-lite stays almost constant for all given arrival rates. Staffware97 is bounded by the number of workflow instances running in parallel. As the execution of invoked applications is carried out on behalf of a single, artificial "dummy" user (called the "auto" user in Staffware terminology), the work queue handling for this user caused congestion, which resulted in long waiting times. The major drawback of *sw-ev* is again its inefficient treatment of data that is returned from the application to the workflow engine.

As the *ml-proc* version of Mentor-lite performed much worse than *ml-thr* in all experiments, this architectural variant is no longer considered in the following experiment.

The impact of the system configuration was evaluated by adding a dedicated computer for the application server. Figure 4 shows the resulting system throughput and workflow turnaround time for the case of short activities with a mean duration of 10 seconds. Mentor-lite is able to fully exploit the additional hardware. Surprisingly, the throughput of Staffware97 dropped compared to the baseline experiment. So for Staffware97 the additional hardware turned out to be a penalty rather than an opportunity. The reason for this seemingly counterintuitive phenomenon lies in the pecularity of Staffware97 that it needs remote-shell calls and thus dynamically spawns remote processes for applications that are invoked on a machine other than the workflow server. The turnaround time for Mentor-lite increases only marginally due to the remote application invocation.
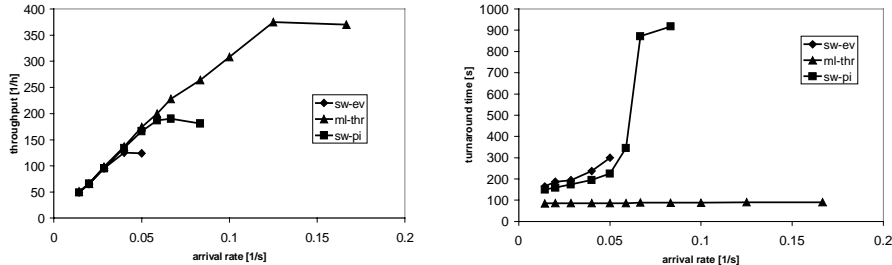

Figure 4 : System configuration impact on throughput and turnaround time

7   Lessons Learned

In addition to demonstrating the viability of the proposed benchmark, the experiments provided a number of important general insights:

- The Corba-based architecture of Mentor-lite proved to be reasonably scalable. In particular and unlike Staffware97, it was able to fully exploit the additional hardware resources of a dedicated application server machine.

- Multi-threading inside the workflow engine turned out to be crucial for bounding the memory consumption of the Mentor-lite workflow server. The version with a separate process for each activity performed very poorly and became unstable for higher load.

- Staffware97 did not scale as well as we expected. First, the file- and event-based version for the communication with invoked applications incurred high delays and resulted in long step response times, critically adding to the workflow turnaround times. Second and even more importantly, the architecture turned out to handle automated activities in a rather inefficient way. Remote invocations on a dedicated application server machine even resulted in decreasing throughput. So Staffware97 obviously is geared mostly for workflows with interactive activities running on client machines, with considerably less stress on the server side.

- Generally, the responsiveness of Staffware97 seems to be critically depending on the configuration of its work queues. For automated activities, all "work items" were assigned to a single queue, for the artificial user "auto", forming a response-time bottleneck. An alternative could be to add more artificial users with some intelligent assignment of activity types to these "auto"-type users, but this would entail a sophisticated load distribution problem and was beyond the scope of our experiments.

8   Architecture of an Auto-Configuration Tool for Mentor-lite

A distributed configuration of Mentor-lite consists of different workflow servers (i.e., instances of the workflow engine), application servers, and one communication server (i.e., ORB). Each server of the first two categories can be dedicated to a specified set of workflow activities and external applications, resp., on a per type basis. Each of these dedicated servers and also the communication server can be replicated across multiple computers for enhanced performance and availability. Given this flexibility (which is provided in similar ways also by some commercial WFMSs), it is a difficult problem to choose an appropriate configuration for the entire WFMS that meets all requirements with regard to throughput, interaction response time, and availability. Moreover, it may be necessary to adapt an initial configuration over time due to changes of the workflow load, e.g., upon adding new workflow types.

To this end, we have developed a suite of analytic models, using stochastic methods like continuous-time Markov chains and Markov reward models, to predict the performance, availability, and performability under a given load. The performance model estimates the maximum sustainable throughput in terms of workflow instances per time unit and the mean waiting time for service requests such as interactions upon starting an activity on the basis of a Markov chain model for the statistical behavior of the various workflow types. The availability model estimates the mean downtime of the entire system for given failure and restart rates for the various components. Finally, the performability model takes into account the performance degradation during transient failures and estimates the effective mean waiting time for service requests with explicit consideration of periods during which only a subset of a server type's replicas are running. These models, which are described in detail in [9], form the underpinnings of an auto-configuration tool for distributed WFMSs.

The auto-configuration tool is primarily driven by statistics on the workload from the monitoring tool of Mentor-lite. It can feed this information into its analytic models to a hypothetical configuration in a what-if analysis. By systematic variation of the parameters for such hypothetical configurations the tool is also able to derive the (analytically) best configuration, i.e., the minimum degree of replication of each of the involved server types to meet given availability and performance or performability goals, and recommend appropriate reconfigurations. The tool is largely independent of a specific WFMS, using product-specific stubs for its various components that need to interact with the WFMS.

The components of the configuration tool and its embedding into the overall system environment are illustrated in Figure 5. The tool consists of four main components: (1) the *mapping* of workflow specifications onto the tool's internal models, (2) the *calibration* of the internal models by means of statistics from monitoring the system, (3) the *evaluation* of the models for given input parameters, and (4) the computation of *recommendations* to system administrators and architects, with regard to specified goals.
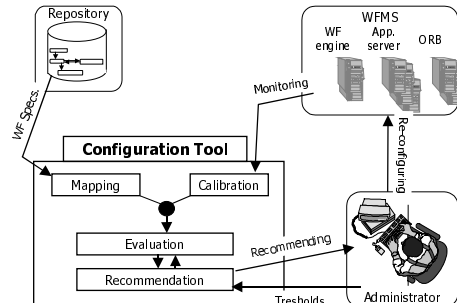
Figure 5 : Integration of the auto-configuration tool

9 Conclusion and Outlook

In this paper we have proposed the first systematic benchmark for WFMS architectures that we are aware of. We have demonstrated the viability and usefulness of the benchmark with measurements of two different WFMSs (each in two different versions). In particular, porting the benchmark, which is initially specified in terms of state and activity charts, to the specification language of Staffware was fairly easy and the installation of the entire setup for the measurements was relatively straightforward (albeit more time-consuming than we expected).

The measured results clearly show that the configuration of a WFMS architecture has a crucial impact on the achieved performance. For example, running an application server on a dedicated machine, as opposed to running it on the same machine as the workflow engine's server, can make a big difference in terms of throughput and turnaround time. To aid system administrators in finding a proper configuration, we have started working on an auto-configuration tool, which we have sketched in the paper. One important issue among the options of the tool is Mentor-lite's capability of having multiple workflow servers on different machines and running even single workflows in a distributed manner. Presumably, high-end commercial workflow systems have similar capabilities or will add such support in the near future. Our future work will include also additional performance measurements for this advanced type of system configuration.

References

[1] G. Alonso, D. Agrawal, A. Abbadi, C. Mohan: Functionality and Limitations of Current Workflow Management Systems, IEEE Expert Vol. 12, No. 5, 1997
[2] T. Bauer, P. Dadam: A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration, IFCIS Int'l Conf. on Cooperative Information Systems (CoopIS), Kiawah Island, South Carolina, 1997
[3] T. Bauer, P. Dadam, Distribution Models for Workflow Management Systems - Classification and Simulation (in German), Technical Report, University of Ulm, Germany, 1999
[4] A. Bonner, A. Shrufi, S. Rozen: LabFlow-1: a Database Benchmark for High-Throughput Workflow Management, Int'l Conf. on Extending Database Technology (EDBT), Avignon, France, 1996
[5] A. Cichoki, A. Helal, M. Rusinkiewicz, D. Woelk: Workflow and Process Automation, Concepts and Technology, Kluwer, 1998
[6] A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.), Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer-Verlag, 1998
[7] A. Geppert, M. Berndtsson, D. Lieuwen, C. Roncancio: Performance Evaluation of Object-Oriented Active Database Management Systems Using the BEAST Benchmark, Theory and Practice of Object Systems (TAPOS), Vol. 4, No. 4, 1998

[8] D. Georgakopoulos, M. Hornick, A. Sheth: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, Vol. 3, No. 2, 1995

[9] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiss: Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems, Int'l Conf. on Extending Database Technology (EDBT), Konstanz, Germany, 2000

[10] J Gray (ed.): The Benchmark Handbook, 2nd Edition, Morgan Kaufmann, 1993

[11] C. Hagen, G. Alonso: Flexible Exception Handling in the OPERA Process Support System, Int'l Conf. on Distributed Computing Systems (ICDCS), Amsterdam, The Netherlands, 1998

[12] D. Harel, State Charts: A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8, 1987

[13] D. Harel, E. Gery: Executable Object Modeling with Statecharts, IEEE Computer, Vol. 30, No. 7, 1997

[14] R. Jain: The Art of Computer Systems Performance Analysis, John Wiley & Sons, 1991

[15] S. Jablonski, C. Bussler: Workflow-Management, Modeling Concepts, Architecture and Implementation, International Thomson Computer Press, 1996

[16] J. Klingemann, J. Waesch, K. Aberer, Deriving Service Models in Cross-Organizational Workflows, 9th Int'l Workshop on Reasearch Issues in Data Engineering (RIDE), Sydney, Australia, 1999

[17] M. Kamath, G. Alonso, R. Günthör, C. Mohan, Providing High Availability in Very Large Workflow Management Systems, 5th Int'l Conf. on Extending Database Technology (EDBT), Avignon, France, 1996

[18] F. Leymann, D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall, 1999

[19] C. Mohan, Workflow Management in the Internet Age, Tutorial, http://www-rodin.inria.fr/~mohan

[20] K. Moore, M. Peterson: A Groupware Benchmark Based on Lotus Notes, Int'l Conf. on Data Engineering (ICDE), New Orleans, Louisiana, 1996

[21] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: Integrating Light-Weight Workflow Management Systems within Existing Business Environments, Int'l Conf. on Data Engineering (ICDE), Sydney, Australia, 1999

[22] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, Enterprise-wide Workflow Management based on State and Activity Charts, in [6]

[23] P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich, G. Weikum: From Centralized Workflow Specification to Distributed Workflow Execution, Intelligent Information Systems, Special Issue on Workflow Management, Vol. 10, No. 2, 1998

[24] M. Nüttgens, T. Feld, V. Zimmermann: Business Process Modeling with EPC and UML: Transformation or Integration, in: M. Schader, A. Korthaus (eds.): The Unified Modeling Language - Technical Aspects and Applications, Workshop des Arbeitskreises "Grundlagen objektorientierter Modellierung" (GROOM), Heidelberg Germany, 1998

[25] P. O'Neil: Database Performance Measurement, in: A.B. Tucker (ed.): The Computer Science and Engineering Handbook, CRC Press, 1997

[26] SAP AG: SAP E-Business Solutions, http://www.sap-ag.de/solutions/technology/index.htm

[27] A.W. Scheer: Benchmarking Business Processes, in: Okino, N.; Tamura, H.; Fujii, S. (eds.): Advances in Production Management Systems, IFIP TC5/WG5.7 Int'l Conf. on Production Management Systems (APMS), Kyoto, Japan, 1996

[28] H. Schuster, J. Neeb, R. Schamburger, A Configuration Management Approach for Large Workflow Management Systems, Int'l Joint Conf. on Work Activities Coordination and Collaboration (WACC), San Francisco, California, 1999

[29] Staffware, http://www.staffware.com

[30] Transaction Processing Performance Council, http://www.tpc.org

[31] TPC-W Benchmark Specification, http://www.tpc.org/wspec.html

[32] Workflow Management Coalition, http://www.wfmc.org

[33] D. Wodtke, G. Weikum, A Formal Foundation For Distributed Workflow Execution Based on State Charts, Int'l Conf. on Database Theory (ICDT), Delphi, Greece, 1997