Workflow Engine Performance Evaluation by a Black-Box Approach

Florian Daniel[1], Giuseppe Pozzi[2], and Ye Zhang[2]

[1] Università degli Studi di Trento, via Sommarive 14 I-38100 Povo, Trento -Italy-
[2] Politecnico di Milano, P.za L. da Vinci 32 I-20133 Milano -Italy-
daniel@disi.unitn.it, http://disi.unitn.it/users/florian.daniel
giuseppe.pozzi@polimi.it, http://home.dei.polimi.it/people/pozzi

Abstract. Workflow Management Systems (WfMSs) are complex software systems that require proper support in terms of WfMS performance. We propose here an approach to obtain some performance measurements for WfMSs (in order to compare them) by adopting a *black box* approach – an aspect that is not yet adequately studied in literature – and report some preliminary results: this allows us to evaluate at run-time the overall performance of a WfMS, comprising all of its constituent elements.
We set up two reference processes and four different experiments, to simulate real circumstances of load, ranging from one process instance to several process instances, entering the system either gradually or simultaneously. We identify some key performance indicators (CPU, main memory and disk workloads, and completion time) for the tests. We choose five WfMSs (some publicly available, some commercially available), and install them in their respective default configuration on five different and separate virtual machines (VMware). For every WfMS and for every experiment, we perform measurements and specifically focus on the completion time. Results enable us to measure how efficient the WfMSs are in general and how well they react to an increase of workload.

Key words: Performance evaluation, Workflow management system, black-box approach, virtual machine

1 Introduction

A *workflow* is the automation of a business process where atomic work units (*task*) are assigned to participants (*agent*) according to a *workflow schema* (*process model*). A workflow management system (WfMS) manages several process instances (*cases*), and relies on a database management system (DBMS).

We propose an approach to evaluate the performances of a WfMS treating it as a *black box* and a monolithic system purely observed from outside. To avoid the variability caused by using different computer systems, we use one unique computer system running several separate virtual machines, each machine featuring one WfMS. The tests we perform particularly aim at measuring the performance of the core of a WfMS, i.e., its engine.

The paper is structured as follows: Section 2 describes the state of the art and compares it with the proposed approach; Section 3 addresses the requirements of our approach; Section 4 recalls some basic concepts on performance evaluation; Sections 5 and 6 introduce our approach and the experiments; Section 7 draws some conclusions and sketches out some future research directions.

2 Related Work

The three main performance-related research areas in the domain focus on:
Impact of WfMSs addresses the changes from the use of a WfMS in managing business processes. Reijers and Van der Aalst [1] focus on analyzing business process performances on the basis of criteria including lead time, waiting time, service time, and usage of resource.
Workflow Process Modeling mostly relates to evaluating the capability of a workflow to meet the requirements of the business process, the workflow patterns [2], adopting as key performance indicators (KPI) maximal parallelism, throughput, service levels, and sensitivity [3]. Several studies focus on the performance issues of process modeling in distributed WfMSs (e.g., Reijers et al. [4] where actors are geographically distributed).
Architectural Issues discuss the inner architecture of WfMSs to improve their performances from the inside. Furthermore, WfMSs must cope with issues such as internet-based large-scale enterprize applications [5], dynamic change of the processes [6]. Kim and Ellis [5] describe three performance analytic models corresponding to the workflow architectural categories of passive, class-active and instance-active WfMS, especially on the aspect of scalability. Considering the configuration of a distributed enterprize-wide WfMS, Kim and Ahn [7] propose the workcase-oriented workflow enactment architecture.

Our Approach. Despite the many previous studies adopt a white box approach and focus on the business layer, on the service layer, or on the internal structure of a WfMS, very few studies address an effective performance evaluation method to assess the WfMS itself as a *black box* on a computer system level, especially on an IT infrastructure layer which is a fundamental component for every system. The *black box* concept here is derived from the *black box* method of software testing, where the inner architecture of programs is not examined. By this paper, we make an effort to design an approach that implements a *black box* performance analysis, to test some WfMSs on an IT infrastructure, thus trying to fill this gap on WfMS performance study.

Our approach builds on the work of Browne [8], evaluating the effective computer performance on the following three factors: 1) theories and models representing computer systems and computer processes; 2) evaluation techniques which generate accurate assessments of the system or of the program behavior from models and theories; 3) the technology for data gathering on executing systems or processes and technology for data analysis. As our performance evaluation is within the context of computer system performance evaluation, we develop three customized factors for WfMS in our approach in the light of [8].

3 Requirements

The paper aims at evaluating the *performance* of WfMSs as it is perceived from the outside, i.e., by system users. The *key idea* to do so is to test each system *as a whole*, instead of singling out individual elements, e.g., the resource scheduler (which decides whether to assign a given task to a human actor or to an automated resource, and to which instance thereof) or the DBMS underlying the WfMS, and testing them individually. Also, in order to test the systems under conditions that are as close to real production environments as possible, we do not want to simulate the functioning of the systems or to emulate individual functionalities. A thorough evaluation requires therefore *setting up full installations* of each WfMS, including the recommended operating system, the DBMS, and any other software required for a standard installation of the systems.

In order to guarantee similar hardware and software conditions for every WfMS, to eliminate the influence ad cross-interactions of already installed software (if the WfMSs are installed on the same machine) or of different hardware configurations (if the WfMSs are installed on different machines), we use a *dedicated virtual machine* for each installation, which can then easily be run and tested also on a single machine. Virtual machines also help the repeatability of the experiments and their portability on other computer systems. We call this requirement *independence from the host system.*

In our research in general, we aim at devising a set of generic performance metrics to assess WfMSs, whereas in this paper – as a first step – we specifically focus on the *workflow engine*, which is at the core of each WfMS and is in charge of advancing process instances according to their *control flows* and managing the necessary *process data* (the process variables). It is therefore necessary to devise a set of *reference processes* we use to study the performance of the engines and minimize the influence of the other components, e.g., the DBMS underlying the WfMS or the resource scheduler. We shall therefore define processes that do not make an intensive use of business data, and formulate tasks that are executed directly by the workflow engine under study. Note, however, that if a WfMS uses a DBMS internally to store its process data, such will be part of the evaluation.

More precisely, it is necessary to define tasks as automatic and self-contained software applications: we call this *autonomy of execution.* In this way, idle times and response times from agents are completely avoided, providing a pure measurement for the WfMS. Every WfMS comes with its own strategies to complete tasks by invoking external software applications: different strategies generate performance variability on both completion time and resource usage, e.g.,to startup suitable applications or persistent variable storage methods.

In order to be able to identify which system performs well in which execution contexts, it is further important to execute the reference processes under *varying workload conditions*. We expect each system will react differently to a growing number of running process instances, concurrently running in the system.

In this work, we are specifically interested in comparing a set of *commercial* and *non-commercial* WfMSs, in order to study if there are differences in how the products of these two families manage their coordination task. Although, at the first glance, the black box approach and the focus on the workflow engine appears to have a certain extent of limitation, we shall see, in the following sections, that this approach already allows us to draw some interesting conclusions on the performance of some state-of-art WfMSs.

4 Background

This section introduces some concepts related to the tests we perform and to key performance indicators (KPIs) for IT infrastructures.

4.1 Performance Testing

Performance testing is one of the computer system performance evaluation methods - also know as Application Performance Management (APM). APM measures the performance and the availability of an application while it is running: the main APM goals are monitoring, alerting, and providing data for incident and capacity management.
Purpose of Performance Testing. Performance testing aims at generating a simulated load to predict the behavior of the system under a real load. During performance testing, we verify what happens if the number of users or of the servers increases (load testing); we also evaluate the capacity of system (stress testing), and find the bottlenecks.
Types of Performance Testing. Two main types of performance testing are typically performed. *Load testing*, also called service level verification, aims at evaluating the behavior of the system under a simulated typical load, in order to verify if the performance goal is fulfilled. During load testing, users enter gradually the system, generating a progressively growing load.

Stress testing is considered a tuning test, as it aims at evaluating the performance of the system under a heavy load to find the maximum load sustainable by every component, helping to detect bottlenecks and providing us with inputs on performance tuning. During stress testing, all the users enter the system simultaneously.
Structure of Performance Testing for an Application. The performance testing structure includes three main components: load generator; controller; monitor. The *load generator* consists of one or more workstations or programs that generate the load on the application. The *controller* is a workstation that controls the load generators to manage the test: it triggers and drives the load generation, by controlling the ramp. The *monitor* is a workstation that measures the system load, performing both generic measurements (e.g., CPU usage) and specific measurements for any component (e.g., application server, DBMS).

4.2 KPI for Performance Testing of an IT infrastructure

KPIs characterize the performance of the measured system and considers several layers of one service. Typically, the most relevant KPIs at the IT infrastructure level consider CPU, CPU idle time for I/O, CPU load (length of the process), main memory usage, disk throughput, and network bandwidth. In Section 5.2 we describe the KPI we shall consider for our goals.

5 Evaluating WfMS Performance

The approach we describe here aims at evaluating the performances of a WfMS. While the literature deeply considers the performances of the single components that set up a WfMS, we propose here a *black box* approach. Our interest mainly focuses on the overall evaluation of the IT infrastructure of a WfMS, as a monolithic system: we do not want to test a WfMS in an isolated fashion, instead we are interested in understanding its performances under real production conditions, that is, also taking into account the minimal system requirements.

While several types of business process can be identified, we introduce two reference processes used throughout the paper. Since we want to test the *pure* performance of a WfMS, any possible human intervention (agent's idle time) and any difference in the computing system must be avoided: all the activities are automatically performed with no human intervention; the same computing configuration and operating condition are used for any WfMS.

The main part of our approach is based on the effective computer performance evaluation framework by Browne [8]. We customize three main factors as follows: *workflow processes*, which are composed by automatic activities; *experimental evaluation procedures*, which generate an assessment of the load for the WfMS under different operational conditions; *performance indicators and data gathering methods*, which include performance measurement factors, tools for performance data gathering and tables for data analysis.

5.1 Workflow Process Design

We introduce two reference processes to evaluate the core behavior of the WfMS: both processes feature a limited set of workflow variables to avoid an intensive use of the underlying DBMS. Consequently, the following processes differ from business processes of the real world.

Although we look at two process models only, we are able to cover a wider set of real-world processes. The first process has a simple structure and a light load; the second one has a more complex structure and an heavier load.

Sample Process #1. The first reference process (SP1) is a very simple, typical, light and effective-running one: tasks are supposed simple, not requiring to execute big software codes. The process includes basic elements and patterns [2], decision nodes, automatic activities, one process variable (i), no temporary variable and no database operation. As the workflow variable can be set up to 1000, the process can cause continuous proper-weight load.

Sample Process #2. The second reference process (SP2) features two parallel execution branches and generates a relatively heavy load. The process contains every kind of routing tasks and patterns [2] (*and-split*, *or-split*, *and-join*, *or-join*),

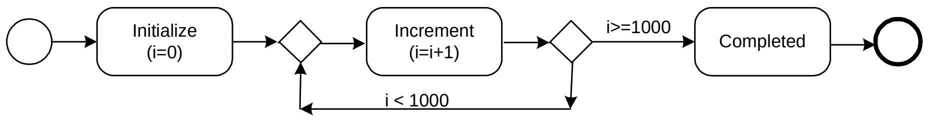

Fig. 1. A simple reference process - SP1.

loops, and wait tasks. The task `Initialize` sets to 0 all the workflow variables i, j, k, m, and n. The task `RandomGenerate` randomly assigns values to a ($a > 0$) and b ($b < 100$): as a and b are randomly generated, every case goes through different execution paths and sets of branches. However, the overall length of the process flow is independent from the values randomly generated for a and b.

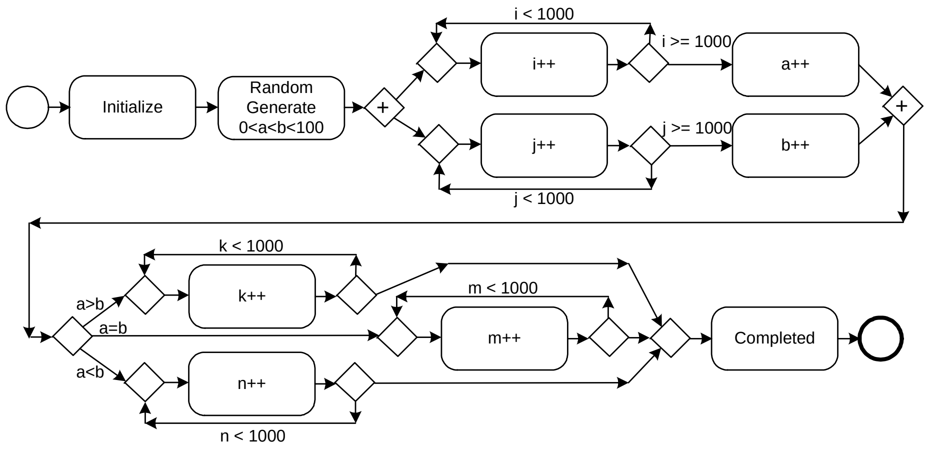

Fig. 2. A more complex reference process - SP2.

5.2 Key Performance Indicators

The suitable KPIs are CPU workload, main memory usage and disk usage. The response time is also an important factor: we consider here the *process completion time*, i.e. the time required to complete the execution of the case.

CPU Workload. Several CPU performance metrics are defined, including CPU usage, system time, user time, idle time and waiting time. As measuring the workload that a certain process generates on the CPU is one of the main goals of our approach, we consider the $ProcessTime/ProcessorTime \times 100$ counter to measure the CPU workload.

Main Memory Workload. Main memory usage is a very critical issue when assessing the performance of a WfMS. Low memory conditions and frequent

page faults slow down the execution of a process. Several performance indicators on main memory workload can be applied: free memory, swap usage and active/inactive memory. We choose *Available MBytes* as a measure of the main memory workload generated by a WfMS during its execution, i.e. the physical memory available, computed as the sum of the zeroed, free, and stand-by memory lists.
Disk Workload. Disk workload monitoring detects how efficiently the application reads/writes data from/to the disk, also exploiting the file-system cache. Considered factors include level of usage, throughput, amount of disk space available. Other activities arise from disk operations, such as interrupts generated by the disk system and paging activity, influencing other resources such as processor or main memory. We consider here the *disk transfer/sec*, as the rate of read and write operations on the disk: it depicts how many I/O operations per second are performed by the guest OS.
Completion Time. This indicator is the average amount of time required to complete a case, measured as the interval between the start time and the completion time of process execution. This is a very direct measurement on how efficiently the WfMS treats the process activity. When running several cases of the same process in parallel, the average process completion time is the interval between the earliest start time and the latest end time of all the cases divided by the number of cases. Consequently, the ability of a WfMS to execute many parallel processes simultaneously will be demonstrated and clarified by the average process completion time.

Observation. Installing several WfMSs on *one* unique system, to have *one* unique physical configuration for the experiments, is a very challenging issue. We use several, separate virtual machines on VMware, each of them running one WfMS, only. Consequently, the measurement of the CPU usage is performed on the *host* OS; main memory and disk workloads, as well as any other parameter, are measured on the *guest* OS. In fact, when we use a virtual machine, we are required to specify disk and memory usage for it. However, we theoretically permit the OS of the virtual machine to use as much CPU as possible. That is to say, we can get quite "comparable" results ("comparable" to the future measurements) by using either host CPU or CPU of the virtual machine. Thus, we use the performance monitoring tool in the host CPU instead of the virtual machine, in order not to put the load of the performance monitoring tool itself into the virtual machine to influence the WfMS.

5.3 Experiment Design

A minimum of *five* runs (Table 1) are executed per experiment and per WfMS. Every run is executed individually, with no overlap, and with no additional program running on the host computer. We set up the following four different types of experiments.
Experiment #1. It measures the performance of a WfMS while one single instance of SP1 is running. This experiment checks the efficiency of the WfMS performing a simple and typical process.

Experiment #2. It measures the performance of a WfMS while one single instance of SP2 is running. This experiment collects data on a WfMS when executing a relatively heavy, all-inclusive process with parallel sub-branches.
Experiment #3. It performs a load testing on the WfMS by SP2, where every *ten* seconds one new instance enters gradually the system till the number of simultaneously running instances is *five*. Theoretically, the maximum number of parallel instances that we should run in this experiment depends on when the workload generated by the current WfMS saturates. The experiment evaluates how the system behaves under the simulated typical operating load of SP2.
Experiment #4. It performs a stress test on the WfMS by SP2, where multiple instances start simultaneously. Experiment #4 can be divided in several parts, according to the number of total running cases: the total number of simultaneously running cases increases progressively, starting from 5 and rising up to 10, 25, 50, 100 or even higher.

Observation. Experiments #3 and #4 differ as instances enter the system gradually (Experiment #3) or simultaneously (Experiment #4). Experiment #4 evaluates the performances under an increasingly heavy load, to find the maximum load of every component and, possibly, the bottleneck.

Physical Configuration. For a fair comparison of the WfMSs, we use *the same* physical configuration for every experiment. We have two physical configurations: the *hardware configuration* includes type of computer/server, CPU, main memory, and hard disk; the *software configuration* includes operating system and the data gathering software, excluding any other user process. However, some WfMSs hold distinct OS compatibility requirements: one of the considered WfMS requires Windows 2003 Server, while the others run on Windows 2000. Windows 2003 Server will produce an additional load on the computer system, somehow affecting the performance evaluation: in order to reduce the impact of the OS, we increase the main memory allocated to balance this additional load.
Sampling Time. The sampling time describes the rate (how often) the monitoring system gathers data from the computer systems. A too high sampling rate introduces an additional and not negligible work load to the computer system itself. In our approach, we choose to acquire samples every 15 seconds, where any observation does not last for more than four hours.

5.4 Data Collection

The performance testing of our approach will produce a large amount of data. We set up some summary tables to collect and analyze results per experiment (Table 1) or per WfMS (Table 2).

For one experiment of one WfMS, the first row of Table 1 describes the considered process (Process template), and the number of cases simultaneously running (# or running instances). Table 1 also depicts all the parameters we collect during every run: the *completion time*, the *CPU workload (%)*, the *available main memory*, the *disk transfer/sec*. To obtain reliable results, we run the same experiment several times (one row of Table 1 for every run, identified by Run Number), computing then the average values for that experiment on that WfMS.

The three resource measurement columns (*CPU workload (%)*, *available main memory*, *disk transfer/sec*) have three sub-columns: *maximum*, *minimum* and *average*. For resource measurements, *maximum* and *minimum* statistics should be taken into consideration because *average* does not suffice to analyze the performances if data change dramatically. *Average* is computed as the mean of all the sampled data. As an example, Table 1 shows that the second run of Experiment #1 on process SP1 for WfMS#1 has a Completion Time of 72,818 msec, the Max value for CPU workload (%) is 10.938, its Min value is 0, its Avg value is 3.125 and so on.

We complete all the runs of Experiment #1 on every WfMS, obtaining 5 instances of Table 1, i.e., one instance per WfMS. We consider the average values (bottommost row) of any table, and put these average values in Table 2, which summarizes the average measurements per WfMS.

If the experiment (e.g., Experiment #3 and Experiment #4) requires several instances running simultaneously, the Total completion time for N simultaneously-started or gradually-entered cases is the interval between the earliest start time and the latest end time of all the processes, while Averaged Process completion time is computed as Total completion time divided by N. Table 3 summarizes the average measurements of Experiment #4 (25 process instances running simultaneously) per WfMS.

Process template - SP1					# of running instances - 1					
Run Number	Single Process Completion Time	CPU workload (%)			Main mem. (MB)			Disk transfer (MB/sec)		
		Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
1	60,656	8.281	0.0	3.182	327	294	308.14	241.195	0.0	37.124
2	72,818	10.938	0.0	3.125	326	293	307.82	209.402	0.0	36.681
3	70,177	9.375	0.0	0.969	327	298	310.71	267.149	0.0	39.836
4	60,103	4.688	0.0	1.133	330	297	312.23	254.382	0.0	38.215
5	64,114	11.208	0.0	2.912	329	299	311.10	173.205	0.0	12.595
average	65,574	8.898	0.0	2.264	327.8	296.2	310.12	229.031	0.0	32.890

Table 1. Measurements for Experiment #1 on WfMS#1. Times are given in msec.

Process template - SP1					# of running instances - 1					
WfMS Number	Single Process Completion Time	CPU workload (%)			Main mem. (MB)			Disk transfer (MB/sec)		
		Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
1	65,574	8.898	0.0	2.264	327.8	296.2	310.12	229.031	0.0	32.890
2	75,457	3.345	0.0	0.661	209.6	182.4	192.596	16.431	0.0	3.043
3	118,928	3.859	0.0	0.358	180.8	175.8	177.857	88.244	0.0	64.231
4	55,755	6.663	0.0	0.812	258.7	256.2	258.236	9.367	0.0	2.478
5	66,650	15.512	0.0	7.643	189.2	109.8	143.972	16.920	0.0	4.098

Table 2. Average measurements for Experiment #1 on every WfMS. Times are given in msec.

Process template - SP2					# of running instances - 25					
WfMS	Averaged	CPU workload (%)			Main mem. (MB)			Disk transfer (MB/sec)		
Number	Completion Time	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
1	28.09	7.187	0.0	1.872	20.0	1.0	2.930	391.191	0.0	251.293
2	36.12	4.192	0.0	0.718	102.0	1.0	31.982	372.124	0.0	41.029
3	22.91	3.981	0.0	0.419	128.0	119.3	120.984	101.923	0.0	57.192
4	32.39	12.981	0.0	2.311	170.6	160.1	168.123	78.837	0.0	8.712
5	23.81	18.939	0.0	7.938	162.0	107.2	114.090	18.571	0.0	6.751

Table 3. Average measurements for Experiment #4 on every WfMS, with 25 process instances simultaneously running. Times are given in minutes - the smaller the better.

6 Experiments and Results

To validate our approach, we apply it to five WfMSs; for every WfMS, we perform the experiments of Section 5.3 and, next, evaluate the obtained results.

6.1 Tool Selection

The WfMSs. In order to obtain a wider scenario of WfMSs, we choose some of the leading, X-PDL-compliant, and open source WfMSs (we name them WfMS#1, WfMS#2, WfMS#4) and some of the top seller commercial WfMSs (we name them WfMS#3, WfMS#5). To avoid any conflict among the installed WfMSs, we install them on five *separate* virtual machines, run them on one unique PC to obtain comparable and hardware independent measurements. The virtual machines run Windows 2000 or Windows 2003 Server, while the host operating system is Windows XP.

The Virtual Machine. As virtual machine, we choose VMware Server rather than VMware ESX server, considering the possible migration of the virtual machine and the potential future extension of our approach.

ESX provides more accurate performance data, as ESX skips the OS level and takes full control of the hardware, directly. On the other hand, VMware Server is somehow influenced by the host OS, like Windows or Linux. However, by taking the direct control of the hardware, ESX has its own file system (VMFS) which provides faster I/O operations but makes it difficult to migrate the virtual machine to other system. Additionally, virtual machine files for VMware Server can be easily stored and transferred to other systems.

Moreover, ESX requires hardware-specific drivers, directly provided by VMware for that selected hardware. Instead, VMware Server clings to the host OS, which provides sufficient drivers to use the virtual machine. This feature makes VMware Server easy to run in a wide range of systems.

Accordingly, we choose VMware Server as our virtual machine environment. We assign to VMware Server an amount of 512 MBytes of main memory and of 8 GBytes of storage: the host machine has 2 GBytes of main memory, 320 GBytes of storage and the Windows XP operating system.

The Performance Monitoring Tool. We adopt the perfmon monitoring tool supported by Windows, collecting data every 15 seconds.

6.2 Experiment Implementation

Given the requirements of Section 3, we implement the autonomy of execution according to every WfMS's own strategy: e.g., in WfMS#1 automatic activities are defined by Java API of J2SE and the GUI structure of the WfMS itself, in WfMS#2 autonomy is achieved by JPDL and Java API.

As for the guest OS, we use Windows 2000 for WfMS#1, WfMS#2, WfMS#3, WfMS#4. We have to use Windows 2003 Server for WfMS#5, due to its strict software requirements (Visual Studio). This requirement influences the overall measurements, due to the different load on the host OS by the guest OS: to partially cope with it, we increase the total amount of main memory assigned to the guest OS to 768 MBytes.

Experiments #3 and #4 require a parallel execution of SP2. The simultaneous activation of several process instances generally differs from WfMS to WfMS: for example, for one WfMS, one process instance can be instantiated via the Web application, but the Web application does not support multiple instantiations. In order to instantiate multiple process simultaneously we program a suitable JUnit code.

We implement the reference processes (SP1 and SP2) on the five WfMSs. We have *five* runs (Table 1) per experiment and per WfMS for experiments #1, #2, and #3; for Experiment #4 we have *five* runs featuring 10 simultaneous instances, and *five* runs featuring 25 simultaneous instances, respectively. The final discussion will consider the average measurements of every set of five runs.

6.3 Analysis of the Results

A huge amount of experimental data support the final discussion and some conclusions on the performances from the five WfMSs. We mainly focus on the process completion time, considered as an overall performance indicator for the *black box* approach we adopt. We do not dig into the internal architecture of the five WfMSs: we rather focus on finding reasons, by combining the external performance data analysis and the general features of WfMSs.

Experiment #1. Experiment #1 features *one* single running instance of SP1. From Figure 3.a, WfMS#4 is the best performer since it requires the minimum amount of time to complete the test. As SP1 is a very simple process, one considerable amount of time is spent during process execution in accessing and updating data in the persistent layer and in loading the program from the logic layer: to our opinion, the great advantage for WfMS#1 comes from using its built-in small DBMS, while other WfMSs use a larger-scale, fully fledged DBMS.

On the other hand, WfMS#3 is the least efficient one for this experiment. To our opinion, this derives from its automatic execution of activities; WfMS#3 requires additional loading time for the activity-relevant executable files deployed

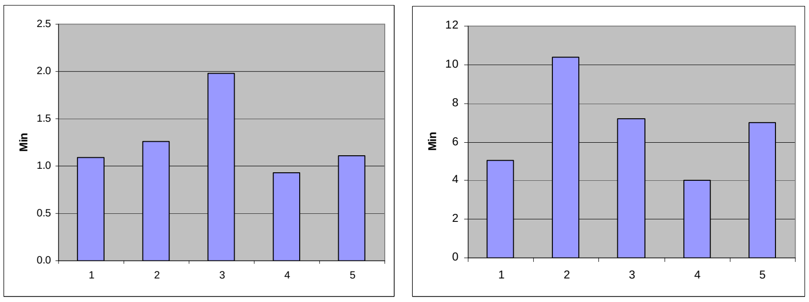

Fig. 3. a: Experiment #1 (left), b: Experiment #2 (right) - average execution time for the five WfMSs (x-axis). Times (y-axis) are given in minutes - the smaller the better

in its server. On the contrary, the execution scripts fulfilling the same functionality in WfMS#1 and WfMS#4 do not require to be loaded because they are located in the process definition file using their self-defined API or script languages.

Experiment #2. As for Experiment #1, Experiment #2 features *one* single running instance of the process, which is now SP2. Comparing Figure 3.a and Figure 3.b, we observe that the efficiency rankings of WfMS#1, WfMS#4, and WfMS#5 remain unaltered. WfMS#2 and WfMS#3 switch their rankings to make WfMS#2 the slowest in executing SP2. Since in SP2 features many routing tasks and parallel task instances, to our opinion the mechanism of WfMS#2 to execute multiple simultaneous task instances is not robust.

Experiment #3. Experiment #3 comes with five instances of SP2 entering the system gradually. The leftmost bars for every WfMS of Figure 4 show that WfMS#1 is apparently the best performer. We identify the following reasons: a relatively small engine with respect to WfMS#3 and WfMS#5, so that WfMS#1 runs the automatic program more quickly without invoking several software layers; the hibernate layer of WfMS#1, which offers an optimized persistent layer solution to facilitate concurrent accesses to the database. This advantage of WfMS#1 is not obvious *a priori*, but shows up when the number of running instances increases within a small range. On the other hand, the process completion times of all the other WfMSs are almost identical within a small variability. This comes from the fact that some WfMSs, which perform fairly well under relatively light load, loose their advantages, while more robust WfMSs gradually catch up on simultaneous execution. Through our experiment results, we identify three of the main factors that have a certain impact on this kind of performance as: the mechanism to execute a small amount of simultaneous process instances; the strategy of space-time trade-off; and the persistent layer robustness.

By comparing Figures 3.a, 3.b, and the leftmost bars of Figure 4, we observe that WfMS#1 is the second fastest WfMS in experiments #1 and #2, and it takes the first place in Experiment #3. Thus, WfMS#1 efficiently executes those process models, both including and not including parallel work items, under a relatively light load (from 1 to 5 instances gradually entering the WfMS).

WfMS#4 is the best performer in experiments #1 and #2, but it is the worst performer in Experiment #3. This could mean that WfMS#4 performs well with *one* single running instance, but its performances fall down rapidly as more instances are executed simultaneously. One reason is that WfMS#4 comes with a persistent layer which interprets and encapsulates data between the logic layer and the persistent layer: consequently, additional time is required to access the database tables when handling simultaneous instances of the same process.

Experiment #4. Experiment #4 comes in two parts, both on SP2: one runs *10* simultaneous instances, the other runs *25*. The middle and rightmost bars of Figure 4 show some reverse trends compared to Figures 3.a, 3.b, and to left bars of Figure 4. WfMS#3 and WfMS#5 execute comparatively slow in experiments #1, #2, and #3 (especially WfMS#3), while WfMS#3 and WfMS#5 are the best performers in Experiment #4, with both 10 and 25 simultaneous instances.

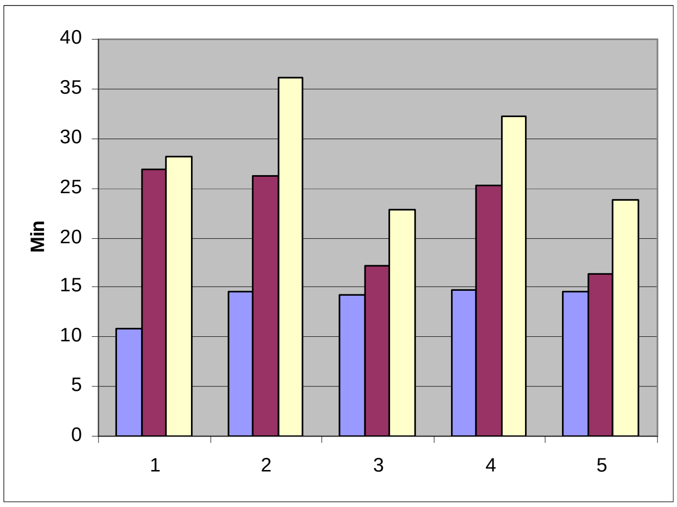

Fig. 4. Average execution time for the five WfMSs (x-axis) for: a: Experiment #3 (leftmost bars for every WfMS) - *five* instances of SP2 gradually enter the WfMS; b: Experiment #4 (middle bars) - *ten* instances of SP2 simultaneously enter the WfMS; c: Experiment #5 (rightmost bars) - *twentyfive* instances of SP2 simultaneously enter the WfMS. Times (y-axis) are given in minutes - the smaller the better.

These behaviors reveal that the two WfMSs show some advantages and robustness when dealing with several simultaneous process instances. For example, WfMS#3 might have a complex persistent layer, which is not so efficient in the simple situation, but is robust and stable as the workload increases. Moreover, some of the good performers under light load circumstances achieve an high execution speed by sacrificing their hardware resources; however, as the workload increases above a (small) threshold, the usage of the resources saturates, and no additional resource can be allocated to further enhance the speed. Therefore, those WfMSs which do not sacrifice their main memory and disk load to obtain a good execution speed, stand out when the number of simultaneous executions increases, despite they are relatively slow under light load circumstances.

7 Conclusions and Future Work

This paper describes a performance evaluation of five state-of-the-art Workflow Management Systems (WfMSs) and the approach we followed in order to compare their performances. We propose a *black box* approach for the performance analysis, as this allows us to study the systems as they are perceived by the actual users of the system, the perspective we are particularly interested in this work. We focus on the workflow engine, i.e., the core of each WfMS, and identify a set of requirements, such as autonomy of execution and independence from the host system, to guarantee a *fair* performance evaluation. The actual evaluation of the systems consists then in three main factors: selection of expressive reference processes; performance data collection; and experiment evaluation. The reference processes feature different levels of complexity; the experiments are defined on the basis of performance testing theory; collected performance indicators include CPU, main memory and disk workload, as well as *process completion time* (our main criterion in this paper).

The conclusions we draw are twofold: (i) under low workload conditions, both commercial and non-commercial products perform similarly; and (ii) under high workload conditions and with more complex process models, commercial products perform slightly better than their non-commercial competitors. Interestingly, this finding is in line with the intuitive decision the typical system designer would take: for less critical and less demanding workloads open-source or free products can suffice, while for mission-critical and demanding workloads a good choice is to invest some money in commercial products – not only to have better product support, but also better performance.

7.1 Future Research Direction

WfMSs are very complex systems and their analysis considers several issues; our approach requires further improvement to become more robust. Firstly, some more typical processes and WfMSs must be considered: next, optimization and fine tuning can significantly modify the behavior of a WfMS – especially for commercial systems, while insofar we just considered the *default* configuration. Further issues concern exception handling, scalability and distributed WfMSs.
Exception Handling. Expected exceptions describe a not negligible semantics of a business process [9]. Whenever a WfMS does not come with an exception handling unit, expected exceptions are mapped within the activity graph of the WfMS [10]. However, such a mapping severely affects the overall performances

of the WfMS: our approach can measure the real incremental workload due to the pure exception management unit.

Scalability. WfMSs are growing up to accommodate an increasing number of users, work items, information items, and complex workflow applications. The architecture of a *robust* WfMS should be scalable up to thousands and millions of instances, with no serious degradation of the performances. Therefore, the total number of simultaneous instances of Experiment #4 is expected to be much larger, while at the same time, WfMSs should be implemented to smoothly operate under heavy load, too.

Distributed WfMS. Throughout the paper we assume that every process instance in a WfMS is executing on one single server. Large enterprizes demand the reliable execution of a wide variety of processes, demanding for a distributed WfMS [11]. Typically, in a distributed WfMS, the process instance is partitioned into several sub-processes, which are executed in a distributed environment on different engines connected via an intranet or even the Internet. Our approach can be enriched to analyze the performances on distributed WfMSs.

References

1. Reijers, H.A., van der Aalst, W.M.P.: The effectiveness of workflow management systems - predictions and lessons learned. In: Journal of Information Management. (2005) 457–471
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14** (2003) 5–51
3. Li, J., Fan, Y., Zhou, M.: Performance modeling and analysis of workflow. IEEE Transactions on Systems, Man, and Cybernetics, Part A **34** (2004) 229–242
4. Reijers, H.A., Song, M., Jeong, B.: On the performance of workflow processes with distributed actors: Does place matter? In Alonso, G., Dadam, P., Rosemann, M., eds.: BPM. Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 32–47
5. Kim, K.H., Ellis, C.A.: Performance analytic models and analyses for workflow architectures. Information Systems Frontiers **3** (2001) 339–355
6. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. Data Knowl. Eng. **24** (1998) 211–238
7. Kim, K.H., Ahn, H.J.: An ejb-based very large scale workflow system and its performance measurement. In Fan, W., Wu, Z., 0001, J.Y., eds.: WAIM. Volume 3739 of Lecture Notes in Computer Science., Springer (2005) 526–537
8. Browne, J.C.: A critical overview of computer performance evaluation. In: ICSE. (1976) 138–145
9. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems. ACM Trans. Database Syst. **24** (1999) 405–451
10. Casati, F., Pozzi, G.: Modeling Exceptional Behaviors in Commercial Workflow Management Systems. In: CoopIS, IEEE Computer Society (1999) 127–138
11. Gillmann, M., Weißenfels, J., Weikum, G., Kraiss, A.: Performance assessment and configuration of enterprise-wide workflow management systems. In Dadam, P., Reichert, M., eds.: Enterprise-wide and Cross-enterprise Workflow Management. Volume 24 of CEUR Workshop Proceedings., CEUR-WS.org (1999) 18–24