An Approach of Performance Evaluation in Authentic Database Applications

Xiaojun Ye, Jingmin Xie, Jianmin Wang, Hao Tang, and Naiqiao Du

Key Laboratory for Information System Security, Ministry of Education
Tsinghua National Laboratory for Information Science and Technology
School of Software, Tsinghua University, Beijing 100084, China
{yexj, jimwang}@tsinghua.edu.cn

Abstract. This paper proposes a benchmark test management framework (BTMF) to simulate realistic database application environments based on TPC benchmarks. BTMF provides configuration parameters for both test system (TS) and system under test (SUT), so a more authentic SUT performance can be obtained by tuning these parameters. We use Petri net and transfer matrix to describe the intricate testing workload characteristics, so configuration parameters for different database applications can easily be determined. We conduct three workload characteristics experiments basing on the TPC-App benchmark to validate the BTMF and the workload modeling approach.

Keywords: Performance testing, benchmarking, test framework.

1 Introduction

In the field of IT systems performance evaluation, testers and hardware/software manufacturers usually focus on different purposes to publish the performance of their IT products with various environments [5]. For example, testers incline to grasp detailed and authentic system status such as its performance, resource utilization or dependability. However, manufacturer's testing purposes are more likely to obtain the performance result of their own product that is comparable with other similar ones. In order to make the evaluation result creditable, tester need to define extremely detailed testing requirements, for example, more authentic business workload, different network delay and every possible think time for each user. It would cost a lot to realize all the detailed requirements modeling in real database systems. On the contrary, some non-profit organizations release performance benchmarks with a high degree of standardization to simplify the testing process and make the performance testing results comparable [12]. These benchmarks not only limit the test database and the database transaction workload, but also the associated performance metrics.

To achieve multiple goals at the same time, emulating test systems, which are close to real database application scenarios and capable to seize comparable performance testing measures, become important for IT system performance evaluation. Two types of approaches are proposed for performance evaluation of emulating systems. One is using general stress testing tools to simulate user requests and responses by invoking

R. Nambiar and M. Poess (Eds.): TPCTC 2009, LNCS 5895, pp. 253–266, 2009.

scripts recorded by testers, and then analyze system performance through the number of concurrent users and maximum throughput [6]. The other type is benchmark testing [12]. Compared with the former, the latter is widely accepted in industry. However, performance benchmarks give too many constraints on the definition of their test database, workload characteristics, performance metrics and SUT (system under test), and benchmark results are rough estimates and only serve the purpose of relative comparison for real systems [10]. To make these components visualized and dynamically configured to satisfy various testing purposes and evaluation targets of different real system characteristic requirements, a domain-independent and model-driven benchmark test management framework (BTMF) emerges from this practice.

Authenticity of simulating different realistic application environments and comparability of various testing results both become important, and this paper aims to find a bridge between them. To better simulate realistic systems based on TPC benchmarks, our BTMF provides configuration parameters in multiple dimensions, so that by tuning these parameters, we can get a more authentic performance result [13]. At the same time, we use Petri net and transfer matrix to describe the intricate and concurrent performance testing workload from business views, various granularity measures and their relationships in real systems under test [3]. With the expandable definition of performance measures [8], customized metrics described by modeling languages, the workload characterization semantics in different test system implementations are explicitly modeled, which is helpful to predict, compare and analyze their corresponding system testing results.

In the next section, we discuss related work of model-driven performance testing and configurable system optimization. In Section 3, the architecture of BTMF is proposed and main components to meet different testing purposes and real system testing environment simulation objectives are detailed. We describe general workload characterization with formal modeling language for simulating different realistic environments in Section 4, and give experimental examples to verify our approach in Section 5. Finally, Section 6 outlines our future considerations.

2 Related Work

Database system performance benchmarking is a well-established area led by Transaction Processing Council (TPC) [12]. With the advance of web technologies and new database application requirements, many benchmarks are updated or replaced in time. For example, TPC-E may supersede the well-known TPC-C lately, and TPC-App derived from TPC-W started to be well accepted by companies [7].

Along with the improvement of performance benchmarks, a variety of performance evaluation methods and techniques ranging from analytical modeling to simulation approaches are designed, including those fault-relevant evaluation methods that focused on specific domains [1, 2], database replay utilities for specific DBMS applications [4]. Therefore, manufacturers need to develop their own performance testing tools for domain-dependent benchmarking, which will add more difficulty for the comparability of the result of test system with other similar products [3, 10].

Literature [10] proposed an application-independent synthetic workload model from the perspective of user's requirements. A high-level specification language, a

translator of the language, and a set of generators were created to compose diverse test databases and test transactions of different synthetic database benchmarks. We learn this model-driven method from the perspective of the user's requirements for test database, workload characterization, workload deployment, and collected measures configuration in the benchmark test management framework.

In performance tuning domain of database application systems, literature [9] proposed an algorithm called Quick Optimization via Guessing (QOG). They formally specified how to guess at the performance and when to terminate measurement, and proved that QOG can find a nearly-best configuration with a high probability under common conditions that are frequently assumed in the literature. The idea of measuring the performance of web systems to optimize configuration parameters can be used in performance optimization area [11, 14]. Hence, our BTMF with flexible configurations are significant during test run since the optimal configuration has been a time-consuming task due to the long measurement time needed to evaluate the performance of a given configuration [13], we propose to use formal language to describe high-level workload characterization in order to predict testing system performance.

Inspired by the model-driven thought and the idea of guessing at the performance in parameter tuning, this paper proposes a configurable BTMF: (1) by means of flexible configurations of the data model, workload characterization and deployment, different granularity measures, this framework can be applied for both benchmark and customized applications performance testing; (2) besides, the approaches of using Petri net or transfer matrix to describe workload configurations (which model real system's business processes), database transactions and performance measures, visualize those intricate relationships and make testing result understandable and comparable in performance metric analysis; (3) since the performance can be guessed based on the similar workload configurations, we can predict the performance of a specific test system configuration according to the formal workload descriptions.

3 Benchmark Test Management Framework

3.1 BTMF Design Philosophy

BTMF architecture consists of a test system (TS) and a system under test (SUT). The TS emulates the user-endpoints which issue requests to the SUT. The SUT in turn responds to these transaction requests. Therefore, we can abstract these components, which are either defined by TPC benchmark standard, or customized by external standards or user-requirements, in BTMF with diverse configuration parameters basing on the model-driven concept. So BTMF components can be customized in these dimensions with predefined parameters for the real application simulation.

In the high-level view of the BTMF, TS includes database manager, workload dispatcher, client emulators and performance measure collector, separately manage test database and generate test data, control user requests dispatching, collect and analyze the performance measures produced by the SUT in terms of the system parameters predefined in BTMF configuration files; SUT includes database, transaction and DB engines in terms of the system parameters predefined in BTMF

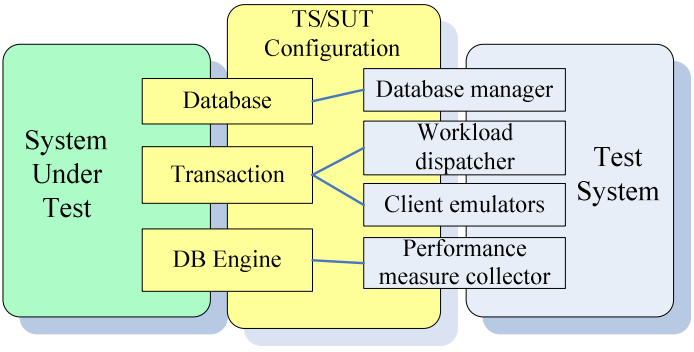

Fig. 1. BTMF design philosophy

configuration files. Therefore, using the model-driven method, we can describe our BTMF design philosophy of SUT and TS as shown in Figure 1.

This architecture is a logical architecture; it does not map functional elements to hardware or software components. Proceeding from these components description to a real IT systems test requires the presence of a complete description of all aspects of the subsystem relevant to the benchmark's performance. This description is called the test system configuration, or the system under test (SUT) configuration.

3.2 BTMF Implementation

The overall components for BTMF implementation include six main modules as shown in Figure 2. *Controller*, *Workload Manager*, *Statistic Collector* and *Data Generator* belong to TS, and *Database* component belong to SUT. *Transactions* component with different granularity measures definition may be in TS or SUT, depending on real system architectures or testing purposes, as we see in different TPC benchmarks. Therefore, BTMF configuration files described by using a high-level language (XML) include configuration parameters for describing workload characteristics, test database, and various measures derived from transactions. These parameters would be analyzed and implemented by the corresponding modules and then be parsed and interpreted by the *Controller* during the testing process.

Data Generator

Data generator mainly has two tasks. One is to model real system data structure by creating divers relations and their semantic restrictions in configuration files, and translate them into real DBMS objects in the form of tables and constraints.

The other task is to generate test database conforming to the data model and data feature definition. We suppose that each independent attribute has a data generation method according to data characteristics predefined in test system (generation rules or user-defined plug-in functions). Like other data generators, BTMF decouples data generation details from user-defined plug-in functions or data generation rules with corresponding configurations in BFM configuration files.

Before populating database, data generator will first analysis table dependence based on foreign key constraints and attribute dependence among attributes of tables

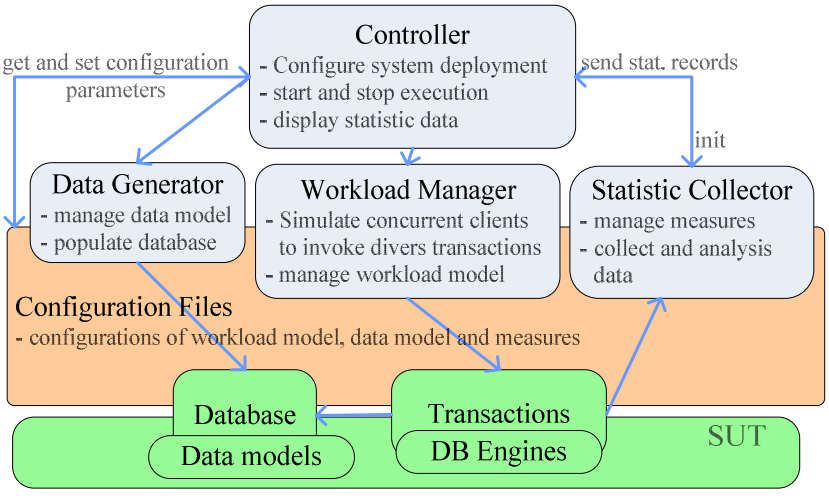

Fig. 2. The benchmark test management framework implementation architecture

based on column level constraints. The connections between output attributes and corresponding input attributes are called as *data dependences*. Before generating test database, the topological structure of these dependences should first be established automatically in order to keep the data semantic. Besides, the acyclic graph of topological structure is divided into several sub-graphs disjointed with each other. Each sub-graph will represent one data generation thread in *Data generator* that BTMF could use to populate table attribute data in order.

The process of database populating has two levels – *table level* and *attribute level*. Taking Figure 3 as an example, A, B, and C are tables; a_i, b_i, and c_i are attributes of tables; the solid and dashed arrows represent dependences between attributes or tables, for example a_1->b_1 means the value of attribute b_1 depends on the value of attribute a_1.

1) Table level: *Data generator* creates the acyclic graph and topological structure of tables, which are listed in Figure 3 (b). Learning from its sub-graphs, BTMF will create two threads to populate data in table A, B and C separately. Considering the broken line in Figure 3 (b), if a_2 depends on b_3, there will be a cycle between table A and table B. In such case, *Data generator* should remove the dependence between a_2 and b_3 first, and then after table A and B is populated, it will recalculate all the values of attribute a_2.
2) Attribute level: When populating data in database tables, the topological structure of all attributes, which represents the order of attributes that the data load module should deal with in each table, should be first established (such as A is listed in Figure 3 (c)). Since there will not be a cycle among these attributes in this example, it is easy for Data generator to populate data in order.

Workload Manager

The workload in database systems can be viewed in two levels. The lower level is manifested by transactions which represent simple business logic unit such as the

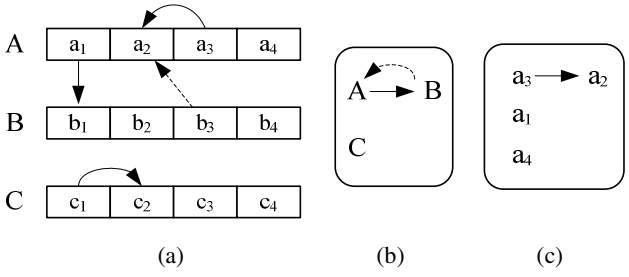

Fig. 3. An example for data generator

"create order" web service in TPC-App benchmark. In this level, the workload is the mixed ratios of transactions, which is an important constraint for TPC benchmark testing requirements. The higher level is manifested by transactional workflows which comprise several tasks with stepwise processes (such as choice, iteration and concurrent execution). The mapping between workflow workloads used in real systems and synthetic transaction workloads used in current benchmarks should be taken into consideration together [3]. By considering more workload characteristics, including the transaction distribution with probabilities, the transaction dependency condition, the input data requirements, etc., the scenarios such as DBMS cache tuning and SQL query optimization during testing process can be more meaningful for real systems performance turning.

As shown in Figure 2, *Workload manager* mainly includes two functions. First, create multi threads to simulate concurrent remote clients to invoke business processes which may be comprised of workflows or transactions. These workflows or transactions are encapsulated in DLL, web service or script according to the realistic environments of simulating systems and the purpose of the comparability of testing results. Second, the mixing ratio of educed database transactions derived from the workflow workload is performed by the *Controller*. These workload characterizations are predefined in configuration files before testing and dynamically invoked by *Controller* during performance evaluation process.

Statistic Collector

During the execution of workflows or transactions, *Statistic collector* will gather performance measures as many as possible, such as begin and end time of a request, submitting number, and throughput. Collected data with the same measure name are connected by a linked list and ordered by submitting time as shown in Figure 4. The linked lists are sorted by hash table. In this way, *Workload dispatcher* can append a line of measure data to the *Statistic collector* and *Controller* can easily get the sorted data from it. With the definition of metrics in configuration files, *Statistic collector* can be also expandable for other specific evaluating purposes.

Statistic collector includes three basic functions: (1) before starting a real test, *Statistic collector* is initialized and ready to receive diverse performance measures from *Client emulators* driven by *Controller*; (2) within an execution, *Workload dispatcher* will add statistic records into *Statistic collector* measure buffer; (3)

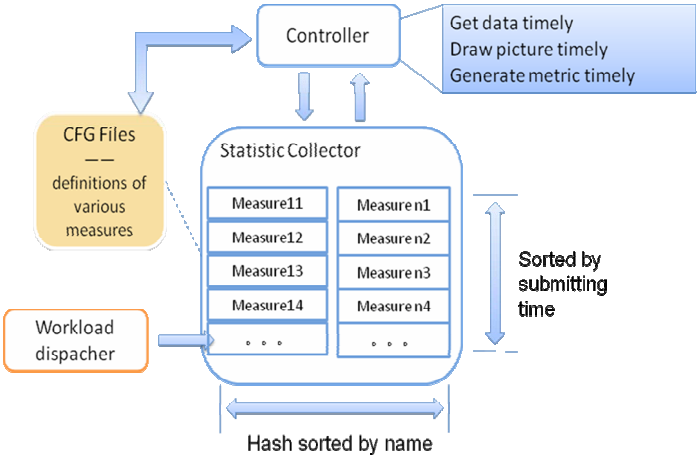

Fig. 4. The store structure and functions of statistic collector

Controller timely gets statistic records from this collector, and then draws performance charts and writes metric data into corresponding BTMF statistic files.

Controller

Controller is used to activate different *Client emulator*'s workloads complying with a fixing ratio from the real system analysis. As listed in Figure 2, *Controller* has three basic functions: (1) getting, parsing and setting configuration parameters predefined by testers in configuration files; (2) starting and stopping data generating and testing execution process, and populating performance measures data in *Statistic collector*; (3) Receiving statistic records, drawing and displaying charts, and writing records into corresponding BTMF statistic files.

The whole testing process of the BTMF includes three steps. First, describe the test database and then populate test data. Second, deploy workload in TS or SUT, design test plan with measures, and initialize *Statistic collector* measure buffer. According to BTMF configuring parameters in configuration files, the *Controller* starts the testing execution and *Statistic collector* records statistic measure data timely.

Transactions, Measures, Database and Configuration Files

Before executing test systems, the transactions and data generating functions should be encapsulated in DLL/web service/scripts programs and their deployed strategies should be described in configuration files in advance. The performance measures are also pre-developed in every workflow/transaction program and predefined in the BTMF functions configuration files.

Apart from workload characterization, data model and feature, performance measures, the configuration items include environment-related parameters (such as database connection string, web service address of *Statistic collector*, store location of statistic files and so on) and testing-related parameters (such as the preheating time of execution, smooth running time etc.). The main objective of configuration files is to build a semantic connection between SUT and TS, and lead our TS to invoke and test

various SUTs with user-defined workload characterization, database models, various performance measures, and system deployment strategies, etc.

4 Workload Modeling

The workload of a database benchmark is the amount of transactions assigned to or performed by a database system in a given period of time. Understanding the nature of the workload and its intrinsic features can help to interpret benchmark performance measures. Transaction dependences are usually overlooked in current OLTP workload modeling. To simulate more realistic of real systems and get a comparable performance result from business views, an authentic and visualized workload would be more helpful. So a formal language is required for keeping the consistence between high-level semantic (workflow) invoked by the simulation client threads and low level transaction mixing ratio in OLTP performance benchmarks. In this section, we illustrate how to use workflow model to describe workload characterization in realistic systems and calculate the mixed ratios of their transactions executed in OLTP performance benchmarks from the high level formal model.

4.1 Simple Workloads in Benchmarks

Though TPC-App replaced TPC-W as the new B2B web service performance benchmark, business transactions, such as "create order", "change payment", and "new customer" transactions, are almost abstracted as database transactions workloads. Recursive calling transactions with mixed ratios (transaction distribution with probabilities) in benchmarks can be abstracted and demonstrated as shown in Figure 5 (a) with Petri net models.

This modeling language provides us a practical view of how to construct and analyze the semantic and similarity of business workloads. For instance, the "create order" web service in TPC-App benchmark will asynchronously send a durable message to shipping process after creating an order in DBMS. Figure 5 (b) gives an abstract of the detailed processes of transaction t_1 with an asynchronous process unit like "create order" transaction, which is often required in current benchmarks.

4.2 Complex Workloads in Realistic Systems

Workflow control patterns are used to better represent business process workloads, while transaction control patterns, where mixing ratio is used to keep the semantic workloads mapping with high-level workflow, are workloads for the current TPC benchmarks. In TPC-W benchmark, transfer matrix is adopted to describe the dependence of transactions. Two other kinds of approaches, by using Markov process and Petri-net, have been brought up to model the relationship among workflows and transactions [3]. With these formal models or languages, the connection of independent transactions in OLTP benchmark with the workflow characterizations to meet the user's real workload modeling requirements can be established. These unambiguous and traceable mathematical descriptions of high-level workload characterization would help testers to calculate the mixed ratios of transactions for emulating database systems and then predict the result semantics of performance of

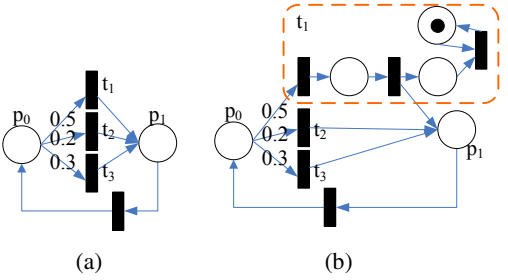

Fig. 5. Examples of simple workloads in performance benchmarks

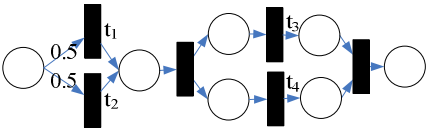

Fig. 6. An example of complex workloads (workflow) in realistic systems

different workflow workloads with the transaction workload for different real systems.

Figure 6 illustrates an example of transactional workflows with a choice of database transaction execution and a concurrent database transaction execution. From the Petri net model, we can calculate that the mixed ratio of transaction t_1, t_2, t_3 and t_4 is 1:1:2:2. In the next section, implementations with different workflows described by Petri net and transfer matrix are tested and analyzed.

4.3 Workload Modeling with Granularity Measures

There are various definitions of the term *performance* in the ISO9126 standard [8]. The most commonly used performance metrics are response time, throughput and utilization. *Response Time* is defined as the time interval between a user request of a service and the response of the system. Some metrics related to response time are *turnaround time*, *reaction time* and *stretch factor* [8]. *Throughput* is defined as the rate at which tasks can be handled by a system, and is measured in tasks per time. For most IT systems, utilization is defined as the ratio of busy time of a resource and the total elapsed time of the measurement period.

In most existing benchmarks, performance metrics are predefined with detailed mathematical formulas, which should not be changed when test systems are developed. In our BTMF implementation, we decide to parameterize these collected measures and use mark transitions from Petri Net to formally denote different granularity measures for business blocks in the workflow model.

Along with the workflows in Figure 6, measures for high level workloads could be added as showing in Figure 7, where three mark transitions for business blocks represented by shadow rectangles are drawn in Petri net graph. We can obtain one performance metric between mark 1 and mark 2, the other one between mark 2 and

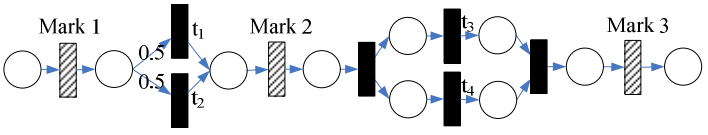

Fig. 7. An example of complex workloads (workflow) with two granularity measures

mark 3. At the appropriate time, these workflow measures data will be sent to *Statistic collector* asynchronously, and high level metrics can be derived timely.

5 Performance Test Result Analysis

In order to validate our BTMF for real database applications, we developed a benchmark test management framework prototype with data models and features, and transaction characteristics derived from TPC-App benchmark [12].

First, we deploy the same database model, transaction characteristics and performance measures as TPC-App benchmark and compare the results with different *active EBs*, *configured EBs* and *mixed ratio of transactions*. Then, based on the transactions of TPC-App, we add another two transaction processes based on Petri net and transfer matrix separately. Through the formal modeling language, we predict the performance results of them and prove them by using real testing results derived from our BTMF implemented prototype tool. Detailed information for database systems testing environment is listed in Table 1.

The test procedure is carried out as follows:

1) Testers perform TS and SUT component configuration, which is the sequence of actions required to perform a benchmark, including TS and SUT software deployment, OS parameter adjustment, etc.

Table 1. The configuration of testing environment

environments		configurations
Test system (TS)	**Controller Machine (Controller and Web Server 1)**	Intel® Core™2 Quad CPU Q6600 2.40GHz
		8G memory, 1T hard disk
		Microsoft Windows Server 2003 R2
		Internet Information Server (IIS) 6.0
	Web Server 1 (for Statistics)	Intel® Core™2 Quad CPU Q6600 2.40GHz
System Under Test (SUT)	**Web Server 2 Machine**	8G memory, 1T hard disk
		Microsoft Windows Server 2003 R2
		Internet Information Server (IIS) 6.0
	Web Server 2 (for Transactions)	Intel® Xeon® CPU E5420 2.50GHz
	Database Server Machine	8G memory, 1T hard disk
		Microsoft Windows Server 2003 R2
		Oracle Database 10g home1 v10.2.0
	Database Server	Microsoft Visual Studio 2005, C#
Platform	**Development Platform**	Microsoft .NET Framework SDK v2.0

2) Test database initialization, in which we use database generator to create test database structure and populate test database according the data characteristics predefined in BTMF configure files.
3) Workload configuration, which is the set of transactions that simulated users database request during test run, together with the relative frequency and relationship with which transactions occur during the test run.
4) Performance test process: obtain a reliable result within an acceptable period.

5.1 BTMF Usability Analysis

The configuration parameters, such as client number, transaction workload, test database model and scale, supporting our BTMF to test diverse scenarios of web database applications, are based on TPC-App benchmark scenarios. Figure 8 shows the result comparison with different active EBs, configured EBs and mixed ratios.

From the left chart of Figure 8 we can find that along with the larger number of active EBs, the value of SIPS/EB metrics (line 'SIPS/EB' and line 'SIPS/EB with different mixed ratios') is smaller and the values of RT metrics (line '90%RT' and line '50%RT') are larger, which means the performance of SUT is lower. At the same time, in the right chart, the performance does not change much along with the larger of configured EBs.

With different mixed ratios of transactions, the performance of SUT may change a lot. The mixed ratios of [new products], [product detail], [new customer], [create order], [order status], [change payment] and [change item] web service transactions are respectively 7:30:1:50:5:5:2 as TPC-App defined and 3:5:10:60:10:10:2 as the author customized, and the performance of them is shown as line 'SIPS/EB' and line 'SIPS/EB with different mixed ratios'. Since the [new customer], [create order], [order status] and [change payment] web services cost more time to be executed than the others, the performance of the latter SUT with user-defined mixed ratios is much lower than the standard mix ratios in TPC-App benchmark.

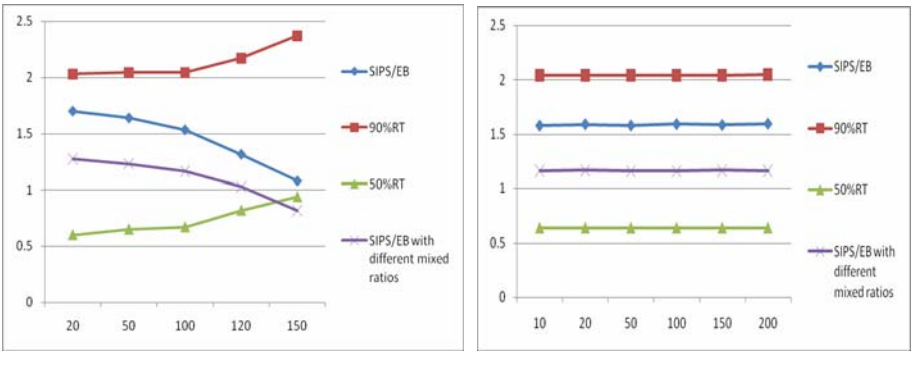

Fig. 8. In the left chart, the x axis represents the number of active EBs, while in the right one the x axis represents the number of configured EBs. The unit of response time (RT) is second.

5.2 Workload Characterization Analysis

In the following, we define three workload scenarios, one is from TPC-App workload model, as shown in Figure 9 (a), one is a transactional workflow with choice and concurrent processes defined by authors described with Petri net as Figure 9 (b), and the third one is a transactional workflow using transfer matrix like Figure 9 (c).

Detailed transfer matrix is shown in Table 2, where the symbol t_1-t_7 represent [new products], [product detail], [new customer], [create order], [order status], [change payment] and [change item] web service in TPC-App. Different workload characterizations, which are represented by Petri net model in Figure 9 (b), transfer matrix model in Figure 9 (c) and Table 2, assures that every web service in each scenarios is still having the same mixed ratios as defined Figure 9 (a). Each value in Table 2 means that when the web service in its row is finished, there will be corresponding possibility to execute the web service in its column, where the blank means that the web service will not be executed after the web service in its row.

We describe these high-level workloads with predefined workflow process definition languages and executed by our BTMF as defined. At the same time, in transaction level, from the mathematic analysis of three types of workloads discussed above (see Figure 9.), we can see that they all have the same mixed ratios of seven types of web services. Table 3 gives the comparable implementation testing results of them. Since they all have the same mixed ratios of the same types of web services, we

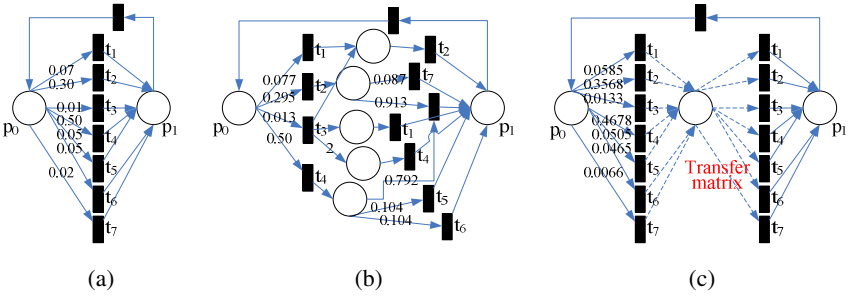

Fig. 9. Three types of workflow workloads with the same mixed ratios of transactions

Table 2. Transfer matrix based on the seven web services in TPC-App benchmark

	t1	t2	t3	t4	t5	t6	t7
t1	0.02	0.02		0.1			
t2				0.3			0.05
t3	0.2	0.3		0.1			
t4	0	0		0.1	0.02	0.03	
t5							
t6	0.2	0.13			0.04		
t7		0.4					
Mixed ratio	0.07	0.30	0.01	0.50	0.05	0.05	0.02

Table 3. Results with different transaction workflows

Workload models	Scenarios in TPC-App benchmark	Scenarios described by Petri net	Scenarios with transfer matrix
Configured EBs	10	10	10
Active EBs	100	100	100
SIPS	157.97	159.19	161.11
SIPS/EB	1.5797	1.5919	1.6111
90%RT(s)	2.04	2.05	2.05
50%RT(s)	0.64	0.64	0.65

can predict that the performance result should be the same or at least very similar. From Table 3 we can see that our prediction comes true, 'SIPS/EB', '90%RT' and '50%RT' metrics are almost the same despite the fact that the implementations have different transactional workflows in high level.

6 Conclusion

We proposed a model-driven benchmark test management framework (BTMF), in which Petri net and transfer matrix are used to describe workload characteristics. The configurable parameters for workload manager, statistic collector, and test database make our BTMF framework applicable for standard benchmarks and authentic applications performance evaluation. The testing results can be predicted according to the mapping of high-level and low-level formal workload descriptions, so the configuration parameters for different database applications can easily be determined.

Both of multiple configuration parameters optimization approaches and workload mathematical modeling with Petri net and other statistical methods will be considered and emphasized in the future. Today's benchmarks do not pay more attentions to the availability issues, such as fault tolerance and recovery cost, thereby, models for performability [8] and analytical method will also be considered together in the BTMS framework.

Acknowledgment

This work was supported by NSFC 60673140 and NHTP (2007AA01Z156, 2008ZX01045-001,2009CB320706).

References

1. Buchacker, K., Tschaeche, O.: TPC Benchmark-c version 5.2 Dependability Benchmark Extensions (2004), http://www3.informatik.uni-erlangen.de/Research/FAUmachine/papers/tpcc-depend.pdf (accessed in July 2009)

2. Costa, D., Rilho, T., Madeira, H.: Joint Evaluation of Performance and Robustness of a COTS DBMS through Fault-Injection. In: The Proc. of DSN 2000, NY, USA (2000)
3. Du, N.Q., Ye, X.J., Wang, J.M.: Toward Workflow-Driven Database System Workload Modeling. In: The Proc. of DBTest 2009, Providence, USA (2009)
4. Galanis, L., et al.: Oracle Database Replay. In: The Proc. of ACM SIGMOD 2008, Vancouver, BC, Canada (2008)
5. Gray, J. (ed.): The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann Publishers, San Francisco (1993)
6. IBM. TPC BenchmarkTM App Full Disclosure Report for IBM® eServerTM xSeries® 366 using Microsoft® .NET 1.1 TPC-App Version 1.1 Submitted for Review (June 21, 2005)
7. HP LoadRunner, `http://www.hp.com` (accessed in July 2009)
8. Koziolek, H.: Introduction to Performance Metrics. In: Eusgeld, I., Freiling, F.C., Reussner, R. (eds.) Dependability Metrics. LNCS, vol. 4909, pp. 199–203. Springer, Heidelberg (2008)
9. Osogami, T., Kato, S.: Optimizing System Configurations Quickly by Guessing at the Performance. In: The Proc. of SIGMETRICS 2007, San Diego, USA (2007)
10. Seng, J.L., Yao, S.B., Hevner, A.R.: Requirements-Driven Database Systems Benchmark Method. Decision Support Systems 38, 629–648 (2005)
11. Swisher, J.R., Jacobson, S.H., Yucesan, E.: Discrete-Event Simulation Optimization Using Ranking, Selection, and Multiple Comparison Procedures: A Survey. ACM Transactions on Modeling and Computer Simulation 13(2), 134–154 (2003)
12. Transaction Processing Performance Council, TPC-C/App/E BENCHMARKTM Standard Specification, `http://www.tpc.org` (accessed in July 2009)
13. Xie, J.M., Ye, X.J.: A Configurable Web Service Performance Testing Framework. In: Proc. of IEEE HPCC 2008, Dalian, China (2008)
14. Zhang, Y., Qu, W., Liu, A.: Automatic Performance Tuning for J2EE Application Server Systems. In: Ngu, A.H.H., et al. (eds.) WISE 2005. LNCS, vol. 3806, pp. 520–527. Springer, Heidelberg (2005)