A simulation process for asynchronous event processing systems: evaluating performance and availability in transaction models

Anakreon Mentis *Panagiotis Katsaros* *Lefteris Angelis*

Department of Informatics

Aristotle University of Thessaloniki

54124 Thessaloniki, Greece

tel.: +30-2310-998532, fax: +30-2310-998419

{anakreon, katsaros, lef}@csd.auth.gr

Abstract

Simulation is essential for understanding the performance and availability behavior of complex systems, but there are significant difficulties when trying to simulate systems with multiple components, which interact with asynchronous communication. A systematic process is needed, in order to cope with the complexity of asynchronous event processing and the failure semantics of the interacting components. We address this problem by introducing an approach that combines *formal techniques* for faithful representation of the complex system effects and a *statistical analysis* for simultaneously studying multiple simulation outcomes, in order to interpret them. Our process has been successfully applied to a synthetic workload for distributed transaction processing. We outline the steps followed towards generating a credible simulation model and subsequently we report and interpret the results of the applied statistical analysis. This serves as a proof of concept that the proposed simulation process can be also effective in other asynchronous system contexts, like for example distributed group communication systems, file systems and so on.

KEYWORDS: asynchronous event processing, simulation, performance evaluation, transaction processing

1. Introduction

Systems with asynchronous event processing involve components that communicate by exchanging messages, which are not delivered in the order that are sent or are even lost. Examples include distributed systems for transaction processing, file systems, data services, group communication services etc. There are two sources of complexity in the specification, design and implementation of these systems: (i) components can be in one of a multitude of states, when they receive a message and (ii) systems involve some failure

semantics. Simulation-based analysis for performance and availability must rely on *verifiably correct models*, as well as on *simulation output analyses* that highlight the main factors that determine whether the anticipated goals can be met.

This work introduces *a coherent simulation process that simplifies the development and analysis of complex models for asynchronous event processing systems*. The process steps include automata-based model specification, model checking, simulation code generation, model validation and simulation output statistical analysis. The whole process has been successfully applied for evaluating the effects of transaction models and their associated parameters on system performance and availability.

Analysis of transaction processing models is supported by the ACID Model Checker [14] and the ACID Sim Tools [16], developed by the authors. Our tool support provides a library of methods for simulating basic operations, such as storing a transaction entry in a log file or attempting a lock on an object. A state machine specification of the system model is checked for the expected correctness properties. The verified model is then used to generate simulation source code that invokes the library methods and manages the model entities interactions. The derived formally-validated simulator is subsequently used in simulations of key execution scenarios (e.g. local to highly distributed transactions, I/O bound or CPU bound resource contention) to find out if the model achieves the expected performance and availability goals. An appropriate statistical analysis explores the feasibility of the performance and availability goals.

The first process steps have been also employed in [18], a related work for a different statistical analysis that specifically focuses on the evaluation of complex metric interactions. In the present article we introduce a complete simulation process that aims to determine whether the anticipated performance and availability goals can be met, rather than a statistical analysis to address existing metric correlations.

Section 2 introduces the steps of the proposed simulation process. Section 3 presents how the process is applied to transaction processing models, as a proof of concept for its effectiveness on complex system models. A synthetic transaction workload scenario is provided to demonstrate the application of the process steps. Section 4 focuses on the statistical analysis of the obtained simulation results towards evaluating the feasibility of expected performance and availability goals. Results for the transaction processing problem of Section 3 are provided. Related work is presented in Section 5 and the paper concludes with remarks on the effectiveness of our proposal and future research directions.

2. Model development and simulation process

Figure 1 presents the simulation process for an asynchronous system model towards evaluating the performance and the availability of a distributed system. The steps of the process are:

1. *Model definition*, by means of state machines specifying the model entities and their interactions. As an example, the classical 2PC transaction model [3] can be defined in terms of states and transitions for the transaction coordinator and worker entities.
2. *Validation for possible discrepancies* between the model and the respective specification, *followed by verification of correctness*. The model is refined until it conforms to the desired *properties*. As an example, the classical 2PC transaction model is expected to "eventually commit or abort all executed transactions" [15].
3. *Generation of event handling code* for simulating the model.
4. *Selection of key execution scenarios* that are either considered as typical use cases, or scenarios that are rarely executed but have a significant performance impact. Typical use cases for transaction models result in scenarios of local or highly

distributed transactions. System failures are rare events, but they can have baneful effects in performance and availability for a transaction model, if the checkpoints (i.e. removal of obsolete entries from the log file) that are taken on a regular basis are not properly configured.

5. For each scenario, *establish performance and availability goals* by specific measurable criteria that an application must meet. Goals are expressed as bounds on metrics such as throughput, response time and operational availability.
6. *Define a workload mix of jobs* arriving with different intensities and the following resource demands:
    a. Software resources such as the number of read/write stable storage accesses, CPU time and number of exchanged messages.
    b. Hardware resources such as the number of CPUs, I/O and network latency.
7. *Validate the developed simulation model* in terms of:
    a. Its utility and effectiveness in identifying the factors that determine the feasibility of the performance goals.
    b. Whether model parameters and their numerical values have real system equivalents.
8. *Execute simulation runs with different combinations of parameter values* for the model. The selected combinations form a designed experiment for exploring the impact of the parameters on performance and availability.
9. *Analyze statistically the simulation results* in order to determine whether the performance goals are achieved. Statistical analysis also reveals the factor(s) that heavily influence goal achievement. If the goals are not met, the found effects guide the analyst into revising the initial model. In transaction processing for example, if a throughput goal fails and an interdependency is identified between

the atomic commit protocol (ACP) and the applied timeout policies, a solution could be to modify the ACP, in order to reduce the time span of exclusive object usage.

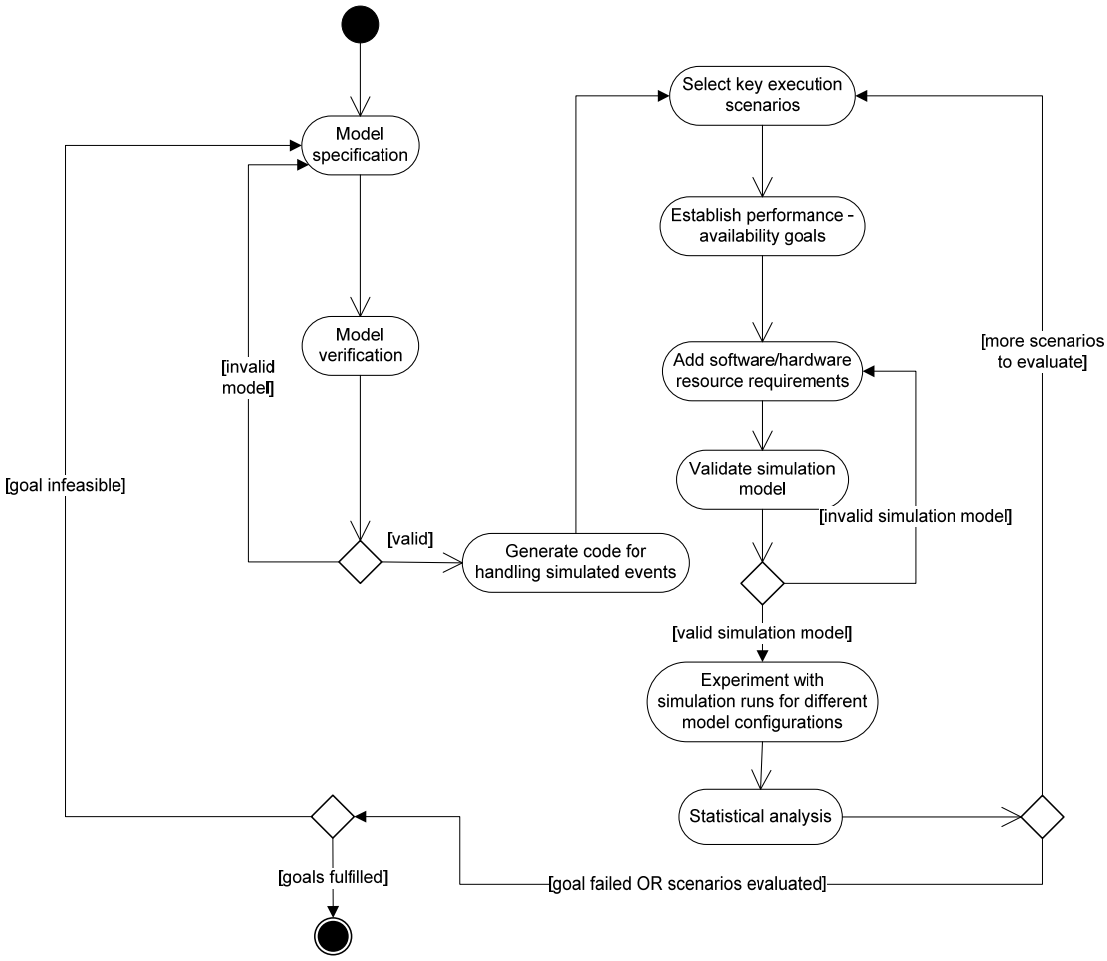


Figure 1. Simulation-based evaluation of performance and availability goals for asynchronous system models

2.1 Model definition

Models are specified in terms of entities that interact with synchronous or asynchronous messages. Most of the model complexity is caused by asynchronous interactions. State machines termed as *roles* define the behavior of entities involved in asynchronous

interactions. Synchronous interactions with the rest of the model entities are encoded in the role state transitions. More specifically, in transaction processing, the transaction coordinator and worker are examples of role entities, whereas the transaction log management and concurrency control functionality is provided by instances of synchronous entities.

State transitions are defined by the source and target states, the triggering message and a sequence of operations invoked upon the state change. The target state of a transition is non-deterministically defined, whenever there is no sufficient information at design time. This is the first source of non-determinism in our modeling approach. Operations may: (i) schedule messages to be received by some entity, (ii) cancel scheduled messages and/or (iii) perform a computation, such as statistics collection. Definition of messages that are generated and/or canceled by all invoked operations is necessary for verifying model correctness. Operations may depend on run-time information to decide which messages are generated or canceled and this is the second source of non-determinism.

2.2 Model verification

In model verification, the analyst provides constraints based on execution history to eliminate the second source of non-determinism. Remaining non-deterministic transitions reflect possible model execution paths that are verified.

During verification, state machines are checked for the possibility of an incomplete transition relation, as well as if the expected model properties are met. An example property considered in [15] for transaction processing is that: “Role entities eventually reach a decision (commit or abort), even in the presence of communication or system failures”. In general, correctness properties are classified in three broad categories, namely safety, reachability and liveness. Verification is performed by an appropriate model checking

algorithm [14]. It is possible that models with many asynchronous interactions cannot be fully verified, unless an advanced abstraction method is applied [5]. This is due to the fact that the algorithm suffers from the state space explosion problem, which is inherent in model checking.

2.3 Code generation for handling simulated events

The model specification that includes roles, state transitions and messages is then used to synthesize simulation source code. The produced code handles the events generated by the role state transitions and the synchronous interactions with the model entities. The only assumption is that an appropriate library provides the implementation of the simulated events. In [16], we provide a library for transaction models that simulates functionality, such as the storage of a transaction entry into a log file, the request for a lock on an object etc.

Code generation is based on an event handling template. Figure 2 shows a simplified version of the used template for transaction model simulation. Placeholders such as <*r*> are replaced by data collected from the model. Comments on the right side indicate blocks of code that are replicated for each element of a collection populated from the model. For example, the code block from lines 8 to 27 is expanded for each role contained in the model, where placeholder <*r*> is replaced by the role name.

The part of the template starting from line 7 is applicable to any asynchronous system simulation. For each role state machine, depending on the message to be handled and the current state, we distinguish between the cases of an enabled deterministic or a non-deterministic transition. In a deterministic transition, the current role state is changed to the target state and the operations associated in model specification with the particular transition are invoked. Non-deterministic transitions (line 18) require user-provided

methods for selecting the transition to be executed, based on run-time information related to the simulated key performance scenarios. As an example from a transaction model simulation, the coordinator, upon a vote received for the transaction outcome, either broadcasts the decision to the transaction workers or waits for the workers that have not yet voted. The number of workers involved in a transaction is specific to the key execution scenarios that refer to particular transaction workloads.

```
1  void <class>::handleTransactionMessage(
     TransactionMessage* msg, const int sndId)
3  {
     int trId = msg->getTransactionId();
5    int msg_type = msg->kind();
     Fsm& fsm = getFsm(trId);
7    switch (fsm.getRole()) {
       case <r>:                    //for each <r> in roles
9      switch(msg_type) {
         case <e>:                  //for each <e> in the events of <r>
11       switch (fsm.getState()) { //for each <s> with a transition for <e>
           case <s>:                //for a deterministic transition
13         fsm.setState(<st>);      //<st> is the transition's target state
           ......                   //specified methods invoked upon a transition
15         break;
           case <s>:                //for a non-deterministic transition
17         //resolve non-determinism at run time
           fsm.setState(resolve<s><e>(fsm));
19         switch (fsm.getState()) {
             case <ss>:             //for each next state
21           ......
             break;
23         }
         }
25       break;
       }
27     break;
     }
29 }
```

Figure 2. Template for event-handling code generation in transaction model simulations

In order to apply this process step to other models apart from transaction processing, the user needs to (i) provide a library with the implementation of simulated events, (ii) modify the lines 1 – 6 of the code generation template and (iii) provide methods that are invoked at runtime to resolve non-determinism.

2.4 Key execution scenarios

Execution scenarios represent workloads of user interactions and the associated system behavior. Key execution scenarios are those, which mainly influence the system's performance from the user's perspective. Two important categories are scenarios that are often executed and scenarios that although rarely executed, have definite performance requirements.

In transaction processing for example, a scenario defines the structure of submitted transactions, i.e. the jobs executed by the transaction and their interdependencies caused by read or read/write access on shared objects. Often executed scenarios include workloads of local read-only transactions. A critical scenario is the recovery after system failure(s), with the requirement that recovery time should be as short as possible. In group communication systems, scenarios define the submitted processes, their submission order and their interdependencies.

2.5 Performance and availability goals

Each key execution scenario is associated with one or more performance goals that establish quantitative criteria for evaluating the system's performance and availability characteristics. For example, a performance goal may state that throughput should be higher than a certain value or it may pose constraints on resource usage (CPU time, number of I/O accesses). Goals are evaluated by metrics, but the used metrics should not mask significant performance variations. This can happen when the model's results are only presented as averages. Simulation allows for inspection of the detailed results (minimum and maximum values), while at the same time appropriate output analysis techniques provide control over the measured variation.

Goals may induce conflicting requirements on a scenario, leading to trade-offs that are addressed in a series of related works. In [10, 11, 17] we have used synthetic metrics that encode the conflicting goals, while in another work [6] we computed cost and benefit metrics, so that different design alternatives can be compared based on their net-benefit value.

2.6 Software and hardware requirements

Software requirements for an execution scenario determine the demands in computational resources, such as the number and size of messages that are exchanged between networked processes, the number of physical I/O operations, the number of CPU instructions etc. Worst case estimates may be used, so that if the goals are addressed it is then unnecessary to invest time into deriving more precise estimates.

Service times for the devices that comprise the execution environment (CPU time, I/O time etc.) are determined based on the estimated software requirements. For instance, the network bandwidth needs to be sufficient for exchanging the expected number of messages, the required CPU time should be available by one or more processors etc. Processing overhead for the exchange of messages, method calls and I/O should be taken into account.

2.7 Simulation model validation

Model validation determines whether the model is an accurate representation of the real-world from the perspective of its intended uses [7]. The model's structure should not contradict the knowledge about the structure of a real system [13] derived either by a specification or experience from a similar system. At the same time, the model's structure should be appropriate, in order to reveal all factors that determine feasibility of the

performance goals. Face validity tests if the abstractions adopted by the simulation expert result in a recognizable representation of the real system.

Finally, simulation parameters (e.g. disk read/write latency and network latency) and their numerical values are examined in parameter-verification tests, if they represent realistic values.

3. Simulation-based evaluation of transaction models

As a proof of concept for our simulation process, we present the evaluation of performance and availability goals for transaction processing systems. Such systems yield complex simulation models with a high degree of asynchronous message processing by the transactional entities. The process steps 1-3 are presented in detail in [14]. Here, we summarize the simulation model abstractions of ACID Sim Tools [16] for transaction processing and we demonstrate the application of process steps 4-8 to a synthetic workload.

3.1 Simulation model abstractions for distributed transaction processing

The adopted simulation model abstractions are based on a minimal set of assumptions that represent an object-based computational model such as the OMG Core Object Model [19]. Clients send transaction requests to application servers that maintain a repository of encapsulated objects. Application servers manage the objects and the runtime environment. Managed objects may invoke methods on other objects residing in the same or in another server.

A transaction is a program execution that reads and/or modifies the state of persistent objects via the invoked object methods [12]. It consists of a sequence of methods executed on one or more servers. Invoked methods incur computations with specific CPU time demands. A transaction is aborted in case of inconsistencies in objects state, potential

deadlocks or overload conditions. Transactions are *local*, if all objects reside in the same server, *read-only* if none of the invoked methods updates the object state and *blocking* if an object is exclusively shared among two or more transactions.

Atomicity, Consistency, Isolation and Durability (ACID) guarantees are provided by the transaction manager. It implements a distributed transaction coordination protocol [24] such as *two-phase commit* (2PC) or one of its optimized variants (e.g. 2PC Presume Commit).

Concurrency control guarantees the isolation property of transactions. Locking protocols require transactions to wait when requested locks cannot be granted. A timeout policy is applied for deadlock resolution, according to which a transaction is aborted when a timer expires. To avoid lock contention thrashing and subsequent deadlocks, transactional servers limit their *multiprogramming level*, i.e. the maximum number of transactions that can be concurrently processed. When this limit is reached, newly arriving transactions are held in a transaction admission queue.

Crash recovery, transaction atomicity and durability are achieved by maintaining recovery data in a *stable log* that survives server crash failures. A server and its objects are unavailable to clients during crash recovery. Recovery time is improved by periodical removal of redundant log entries (checkpoints).

3.2 Key execution scenarios

Key execution scenarios include workloads with various degrees of distribution (localized to highly distributed transactions), different mixes of read-only, update and blocking transactions that affect lock contention and diverse conditions of resource contention, such as I/O bound and CPU bound workloads. Scenarios that occur rarely, but have a significant

performance effect include settings with different frequencies of server failures and checkpoint intervals.

3.3 Performance and availability goals

Typical performance goals for distributed transaction processing concern throughput, response times and operational availability. Throughput is the percentage of committed transactions. Response time is the time span between issuing a transaction request and committing the transaction. Availability is the percentage of time during which a transactional server processes incoming requests. Performance goals are often connected with a particular transaction type, such as the distributed or the locally executed transactions or even a group of transactions that share common locks. For this reason, we advocate the use of separate throughput and response time metrics for the mentioned transaction types, since they can exhibit very different performance behavior in a system.

High availability requires short recovery times after server failures. The size of the transaction log file is important not only for the recovery costs, but it also influences rollbacks of aborted transactions: locks are retained until transaction recovery is complete. Transactions that await processing are blocked until the used locks are released, thus affecting the measured throughput. It is therefore evident that throughput and response times depend on the implementation, the configuration and the tuning of the transactional servers.

3.4 Synthetic transaction workload

The studied key execution scenarios are based on a synthetic transaction workload that uses a number of objects distributed among two servers (`acp1` and `acp2`). Table 1 summarizes the aggregate computational and memory requirements for the transaction classes that

comprise the workload. Transaction classes invoke object methods in a specific order. Lock acquisition on the shared objects is performed in the same order. We assume exponentially distributed size of object states and CPU time demands for the invoked methods and we show the aggregated means over all objects that reside on the same server.

Table 1: Transaction classes (CPU time and memory demands)

Transaction Class	Characteristics	CPU demands (sec) - exponential -		State Size (Kb) - exponential -	
		acp1	acp2	acp1	acp2
tr1	local (acp1), read – only	0.03		15	
tr2	distributed, read – only	0.06	0.01	10	5
tr3	distributed, read – write	0.06	0.05	10	5
tr4	local (acp2), read – only		0.07		15
tr5	distributed, read – only	0.01	0.1	5	5
tr6	distributed, read – write	0.01	0.1	5	10
tr7	local (acp2), read – only		0.07		15
tr8	distributed, read – only	0.01	0.02	5	5
tr9	distributed, read – write	0.01	0.06	5	10

Table 2: System parameters

Network Latency / message:	0.06 sec		
Server	Disk Read Latency	Disk Write Latency	Mean Time To Repair
acp1	4.271e-05 sec/Kb	51.252e-05 sec/kb	4 sec
acp2	4.271e-05 sec/Kb	51.252e-05 sec/kb	4 sec

System-related parameters are summarized in Table 2. We assume fixed network latency, since the two servers exchange only small-size control messages. Message losses can be easily modeled in ACID Sim Tools by assigning a loss probability in the communication of the transactional servers.

The basic two phase commit protocol and the two implemented variants are blocking protocols: upon a network failure, objects that participate in transactions are either aborted or, if the transaction has entered the voting phase, all participating objects are locked until the network link is restored. Occasional message losses are therefore tolerated, but we do not expect significantly different effects in the system's performance. For this reason, message losses have not been included in the simulated synthetic workload.

3.5 Simulation model validation

A series of parameter verification tests build our confidence that the used parameter values represent real system cases. Network latency, for example, depends on the network infrastructure and the distance between the communicating peers. Measurements taken in our experiments yielded values averaged over 60 ms, which is in agreement with latency statistics provided by network vendors [25]. Read and write disk latency is obtained by vendor specifications.

3.6 Simulation experiments

Let us consider the following performance goal:

"Optimize the throughput of the locally executed transactions that form the larger part of the expected workload"

This goal may be connected to some lower bound for the throughput of the locally executed transactions.

Table 3: Parameters for key execution scenarios (factors)

Factors	**Level 1**	**Level 2**	**Level 3**
Atomic commit protocol (ACP) - all servers -	Two-Phase Commit Presume Nothing (PRN)	Two-Phase Commit Presume Commit (PRC)	Two-Phase Commit Presume Abort (PRA)
Multiprogramming Level (MPL) - all servers -	2	3	4
Checkpoint intervals (CI) - periodic (sec) for all servers	500	1300	2100
Transaction timeouts (TT) in sec - all transaction classes -	0.9	1.1	1.3
Mean interarrival times (MIT), exp. - all transaction classes -	0.4	0.6	
Mean interarrival time of server fail -stop failures (MITofSF) - exponential	18 m	5 hours 51m	12 hours

The research question associated with the aforementioned goal is:

"How throughput of locally executed transactions and other metrics are affected

by the statistically significant factors in the considered execution scenarios"

Systematic experimentation and statistical analysis can indicate the required parameter values, so that the expected goal is fulfilled, if it is feasible. Experimental design was based on preliminary simulation runs for exploring the variability of the performance metrics and parameters were assigned values, so that the experimental region covers this variability as much as possible. A full experiment was performed with all combinations of the parameter values shown in Table 3. In all simulation runs, concurrency control adheres to the widely used two-phase locking (2PL) scheme.

The full experiment includes 486 cases with simulated time of 55h 30m for each of them. Sufficiently long simulation runs ensure that results are obtained when the model is in the steady-state. The CPU time in a personal computer with a single-core processor and 1 GB RAM varied between a few minutes to 12 minutes, depending on the used simulation parameters. Experiments with short interarrival rate and high multiprogramming level process more transactions and require more time to complete. ACID Sim Tools exhibited a stable behavior without memory leaks and the memory usage varied between 5 % and 50 % of the system's memory. Alternative experiment designs with less CPU time are the so-called uniform designs that we used in [10]. For each simulation case, the method of independent replications was used, in order to compute 95% confidence intervals for the metrics of interest.

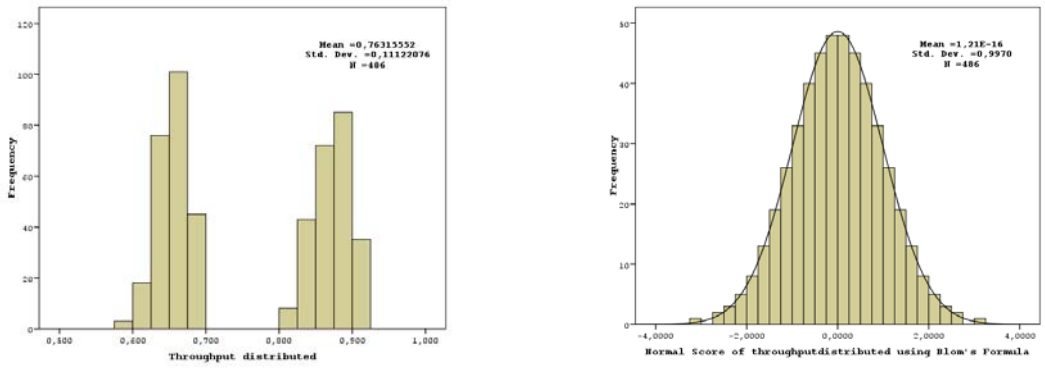
4. Statistical analysis

We propose the use of multivariate analysis of variance (MANOVA) [8], which is an extension of factorial ANOVA *designed to simultaneously look at several outcomes of an experiment*. MANOVA is essentially a multivariate test that limits the inflation of the error rate resulting from multiple tests (univariate ANOVA), where each test examines a

dependent variable (metric) [9]. MANOVA also detects groups of interacting dependent and independent variables by examining their correlations.

In the simulation results for the considered key execution scenarios, we realized that all histograms for the measured metrics exhibit either high skewness or multimodality (different distributions), which means that they are not normally distributed. Since the analysis of variance requires the dependent variables to be normally distributed, we applied the Blom's transformation [4] that is based on the rankings of the values in each variable. This transformation utilizes the ranks r_i of the values and the cumulative Normal distribution function $\Phi^{-1}(.)$:

$$s_i = \Phi^{-1}\left(\frac{r_i - 3/8}{n + 1/4}\right)$$



(a) (b)

Figure 3. Distribution of simulation results for the throughput of distributed transactions before and after the normal transformation

The histogram in Figure 3a shows the distribution of the values representing the throughput of distributed transactions in the performed simulation runs. Figure 3b shows that the distribution of the new variable is almost perfectly fitted by the standard normal distribution with mean 0 and standard deviation 1. The selected transformation preserves the effect of the factors (Table 3) on the dependent variables (metrics).

MANOVA is simultaneously applied to all normal transformations of the dependent variables, with respect to the 6 factors. The model includes initially the 6 main factor effects, the 15 two-way interactions (i.e. all possible combinations for unordered pairs of factors) and the 20 three-way interactions. Interactions of higher order are not considered, since it would complicate the interpretation of the results.

By examining the F-ratios produced by MANOVA, we concluded that the following interactions are not significant:

ACP * MPL * CI, ACP * MPL * MITofSF, ACP * CI * TT, ACP * CI * MIT, ACP * CI * MITofSF, MPL * CI * TT, CI * TT * MIT, ACP * MPL

All main effects and the rest of the factor interactions are statistically significant for the obtained model. Regarding the validity of the model, it is well fitted to the simulation output: the R-squared value for each dependent variable that shows the percentage of variability explained by the model is in all cases very close to 1.

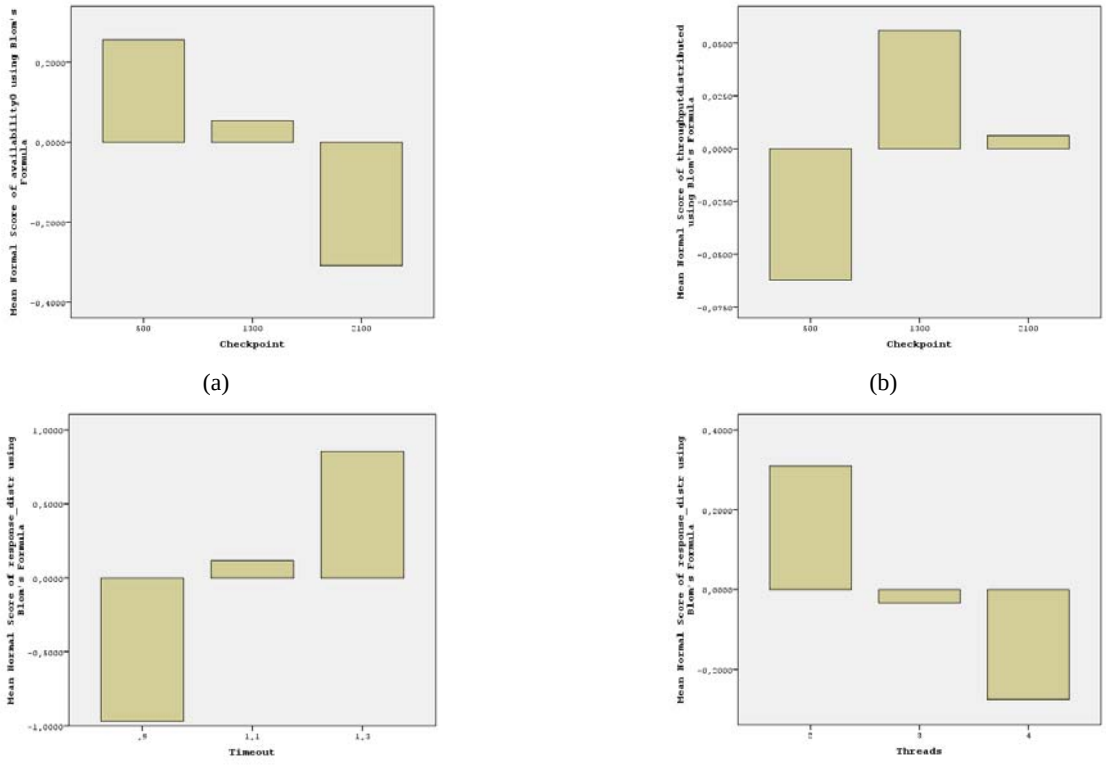
In order to assess the impact of each factor on the dependent variables, we performed a series of statistical tests that accompany the MANOVA implementation in the statistical program SPSS [22]. Table 4 summarizes the factors that have statistically significant effect on the dependent variables.

Table 4: Main factor effects on transaction processing performance and availability
(X = statistically significant effect, – = no effect)

Metrics	ACP	MPL	CI	TT	MIT	MITofSF
Acp1 server availability	-	-	X	-	-	X
Acp2 server availability	-	-	X	-	-	X
Local transactions throughput	X	X	X	X	X	X
Distributed transactions throughput	X	X	X	X	X	X
Mean response time for local transactions	X	X	X	X	X	X
Mean response time for distributed transactions	X	X	X	X	X	X
Mean blocking time	X	-	X	X	-	X

In Figure 4 we provide representative bar-charts that are indicative of the way that dependent variables are affected by the considered factors. The height of each bar shows the transformed mean value for a specific factor level.

Figures 4a and 4b show the impact of checkpoint interval on the availability of the Acp1 server and the throughput for distributed transactions. Frequent checkpoints improve the measured server availability by reducing the size of the log file. On the other hand, checkpoints compete with the ongoing transactions for I/O and prolong the recovery of aborted transactions. In effect, frequent checkpoints cause degradation of the measured throughput (Figure 4b).



(a) (b)

(c) (d)

Figure 4. Bar-charts for the main factor effects on performance and availability

Figure 4c shows the impact of timeout intervals on the throughput of distributed transactions. More transactions can commit for longer timeouts although other performance goals such as response time deteriorate. In Figure 4d we see that the multiprogramming level improves the response time of distributed transactions. With more available threads, transactions spend less time queued for computational resources.

Profile plots like the ones shown in Figure 5 are appropriate for visualizing the interaction effects between factors. They are an important evaluation tool for the impact of different combinations of protocols and protocol parameters on performance goals. For the sake of brevity we present only a few of the produced profile plots.

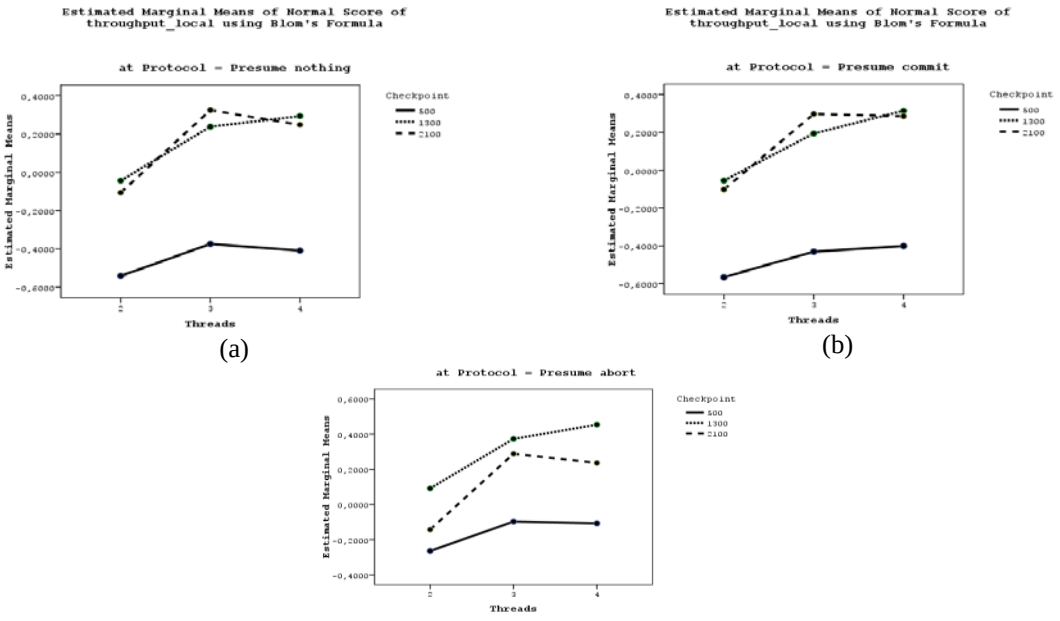


Figure 5. Interaction effects between CIs, MPLs and ACPs on the throughput of local transactions

Figure 5 reveals an interesting interaction effect between the ACPs and the CIs, as well as between the CIs and the MPLs on the throughput of local transactions. A significant improvement is observed for CIs period of 1300 sec and the 2PC Presume Abort protocol.

We also note differences in the rates of variation for the examined CIs intervals, when increasing the servers' MPLs.

Regarding the performance goal of section 3.6 we observe that:

"The best options for the throughput of locally executed transactions, is the combination of 2PC Presume Abort with four available threads in the system's servers and CIs that do not diverge significantly from the level of 1300 sec."

Figure 6 visualizes the interaction effects between CIs, MPLs and ACPs on the availability of the Acp2 server. We observe that when the checkpoint frequency is overly high, using more than three threads has a negative effect on the measured availability. 2PC Presume Abort seems to scale evenly for MPLs from 2 to 4 threads for not overly frequent CIs.

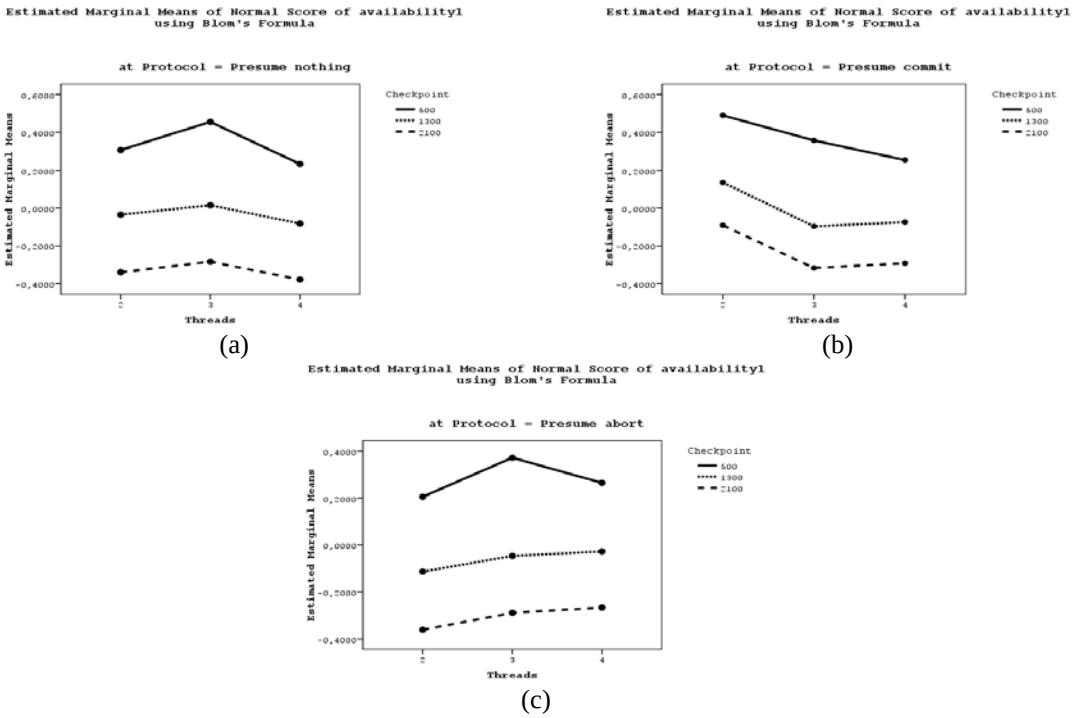


Figure 6. Interaction effects between CIs, MPLs and ACPs on the availability of Acp2

5. Related work

Our approach targets the problem of simulation-based performance analysis for asynchronous systems.

A detailed account of existing model checking techniques for software architecture specifications is provided in [26]. There are also several works focusing on model driven development for simulation analysis. In [21], the authors propose a systematic performance evaluation process based on queuing networks. Models are assumed correct, which may be not self-evident for asynchronous systems with complex interactions between the model components.

It is true that very few works can generate verifiably correct simulation models. The BIP (Behaviour, Interaction, Priority) component framework [2] introduces atomic components, expressed by state machines, that interact via connectors. Selection between simultaneously enabled interactions is performed based on their assigned priorities. A hierarchy of atomic and compound components is used to model the target system [23] and to generate a concrete implementation. Correctness with respect to system properties is derived out of correctness properties local to the atomic components. BIP also features a simulation-based performance analysis tool. In overall, BIP mainly focuses on the development of correct-by-construction executable models, whereas our approach is based exclusively on model checking. An additional characteristic of our work is the proposal for multivariate analysis of the simulation results.

Timed Rebeca [1] is an actor-based modelling and simulation language. Model checking is performed on a mapping of Rebeca models into timed automata and the produced models can then be simulated. The simulation results are analyzed, in order to determine if the performance goals are achieved. The authors do not consider a systematic performance analysis process like the one we propose with the multivariate statistical analysis.

6. Conclusion

We introduced a simulation-based process for the performance analysis of asynchronous event processing systems. Our process covers the problems of model specification, model checking for the generation of correct simulation code, selection of key execution scenarios for the performance goals of interest, model validation and simulation output statistical analysis.

The process has been successfully applied on a synthetic workload with key execution scenarios of distributed transaction processing. This is a typical case of complex systems with asynchronous event processing. We tackle the complexity associated with the performance evaluation of such systems, based on (i) “correct” simulation models for faithful representation of the complex system effects and (ii) statistical analysis for simultaneously studying several outcomes of the simulation experiment.

As a future research prospect, we plan to extend our approach towards the development of a performance benchmarking methodology by applying advanced statistical techniques of archetypal analysis [20].

References

[1] Aceto, L., Cimini, M., Reynisson, A. H., Ingolfsdottir, A., Sigurdarson, S. H., Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca, Proc. 10th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA), Aachen, Germany, pp. 1-19, 2011

[2] Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T., Sifakis, J., Rigorous Component-Based System Design Using the BIP Framework, IEEE Software, 28 (3), pp. 41–48, 2011

[3] Bernstein, P. A., Newcomer, E., Principles of Transaction Processing, 2nd Ed., Morgan Kaufmann, 2009

[4] Blom, G., Statistical estimates and transformed beta variables, Wiley, New York, 1958

[5] Clarke, E. M., Grumberg, O., Peled, D. A., Model Checking, MIT Press, 1999

[6] Deshpande, T., Katsaros, P., Basagiannis, S., Smolka, S., "Formal analysis of the DNS Bandwidth Amplification Attack and its countermeasures using probabilistic model checking", Proc. 13th IEEE Int. High Assurance Systems Engineering Symposium (HASE), Florida, pp. 360-367, 2011

[7] DoD, DoD Instruction 5000.61: Modeling and Simulation (MandS) Verification, Validation, and Accreditation (VVandA), Defense Modeling and Simulation Office, Office of the Director of Defense Research and Engr., 2002

[8] Harris, R. J., A primer of multivariate statistics, 3rd Edition, Lawrence Erlbaum Associates, Mahwah, 2001

[9] Katsaros, P., Angelis, E., Lazos, C., "Applied multiresponse metamodeling for queuing network simulation experiments: problems and perspectives", Proc. of the EUROSIM 2001 Congress, EUROSIM, Delfts, The Netherlands, 2001

[10] Katsaros, P., Angelis, L., Lazos, C., "Performance and effectiveness trade-off for checkpointing in fault tolerant distributed systems", Concurrency and Computation: Practice and Experience, 19 (1), John Wiley & Sons, pp. 37-63, 2007

[11] Katsaros, P., Lazos, C., "Optimal object state transfer - recovery policies for fault tolerant distributed systems", Proc. of the IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN), Florence, Italy, pp. 762-771, 2004

[12] Martin, C. P., Ramamritham, K., "Toward formalizing recovery of (advanced) transactions", in: Jajodia, L. Kerschberg (Eds.), Advanced Transaction Models and Architectures, Kluwer, Boston, 1997

[13] Martis, M. S., “Validation of simulation based models: a theoretical outlook”, The Electronic Journal of Business Research Methods, 4 (1), pp. 39–46, 2006

[14] Mentis, A., Katsaros, P., “Model checking and code generation for transaction processing software”, Concurrency and Computation: Practice and Experience (to appear), John Wiley & Sons, 2011

[15] Mentis, A., Katsaros, P., “The ACID model checker and code generator for transaction processing”, Proc. of the 2009 High Performance Computing & Simulation Conference (HPCS), pp. 138–144, 2009

[16] Mentis, A., Katsaros, P., Angelis, L., “ACID Sim Tools: A simulation framework for distributed transaction processing architectures”, Proc. of the 1st Int. Conference on Simulation Tools and Techniques for Communications, Networks and Systems, Marseille, France, 2008

[17] Mentis, A., Katsaros, P., Angelis, L., “Synthetic metrics for evaluating performance of software architectures with complex tradeoffs”, Proc. of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Patra, Greece, pp. 237-242, 2009

[18] Mentis, A., Katsaros, P., Angelis, L., Kakarontzas, G., “Quantification of interacting runtime qualities in software architectures: insights from transaction processing in client-server architectures”, Information and Software Technology, 52 (12), pp. 1331-1345, 2010

[19] Object Management Group, Object Management Architecture Guide, revision 3.0, OMG Technical Committee Document ab/97-05-05, June 1995

[20] Porzio, G. C., Ragozini, G., Vistocco, D., “On the use of archetypes as benchmarks”, Applied Stochastic Models in Business and Industry, 24, pp. 419-437, 2008

[21] Smith, C. U., Williams, L. G., Performance solutions: A practical guide to creating responsive, scalable software, Addison-Wesley, 2001

[22] SPSS for Windows, SPPS Inc, http://www.spss.com, (last access: 23rd of April 2012)

[23] Stachtiari, E., Mentis, A., Katsaros, P. "Rigorous analysis of service composability by embedding WS-BPEL into the BIP component framework", Proc. of the 2012 IEEE 19th International Conference on Web Services (ICWS), Honolulu, Hawaii, pp. 319-326, 2012

[24] Thanisch, P., "Atomic commit in concurrent computing", IEEE Concurrency, pp. 34-41, Oct. – Dec. 2000

[25] Verizon network latency statistics, Verizon, http://verizonbusiness.com/about/network/latency, (last access: 23rd of April 2012)

[26] Zhang, P., Muccini, H., Li, B., "A classification and comparison of model checking software architecture techniques", Journal of Systems and Software, 83 (5), pp. 723-744, 2010