Synthetic metrics for evaluating runtime quality of software architectures with complex tradeoffs

Anakreon Mentis Panagiotis Katsaros Lefteris Angelis
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{anakreon, katsaros, lef}@csd.auth.gr

***Abstract*—Runtime quality of software, such as availability and throughput, depends on architectural factors and execution environment characteristics (e.g. CPU speed, network latency). Although the specific properties of the underlying execution environment are unknown at design time, the software architecture can be used to assess the inherent impact of the adopted design decisions on runtime quality. However, the design decisions that arise in complex software architectures exhibit non trivial interdependences. This work introduces an approach that discovers the most influential factors, by exploiting the correlation structure of the analyzed metrics via factor analysis of simulation data. A synthetic performance metric is constructed for each group of correlated metrics. The variability of these metrics summarizes the combined factor effects hence it is easier to assess the impact of the analyzed architecture decisions on the runtime quality. The approach is applied on experimental results obtained with the ACID Sim Tools framework for simulating transaction processing architectures.**

Keywords-software architecture; runtime quality; architecture design tradeoffs

I. INTRODUCTION

Software architecture is defined as "*the structure or structures of the system, which include software components, the externally visible properties of those components, and the relationships among them*" [2]. For the software, the implemented architecture design sets the boundaries for its runtime quality, which is quantified by a number of metrics, where each metric reflects a different aspect of runtime behavior. The architecture cannot be a basis for precise predictions of the system's runtime behavior, since it also depends on implementation details. However, the architecture can support analyses that provide some level of confidence for the main effects of its characteristics, called *factors*, that affect the runtime metrics of interest.

Software designers are aware of the factors affecting runtime quality. However, predicting the most influential factors is not straightforward or obvious [6]. The effect of a single change in a factor of the software architecture is likely to affect multiple metrics of runtime quality. For example checkpoint is an architectural factor in a transaction processing system that contributes in achieving high availability, at the cost of potential deterioration of the system's performance. This is a typical runtime quality tradeoff.

We propose the construction of a few synthetic metrics that summarize the effects of architectural factors, by taking into account the correlations between the metrics of interest.

The approach employs a statistical analysis, which may be applied either on simulation data or monitoring data obtained from a prototype. This analysis discovers groups of runtime metrics which are strongly correlated, because they are found to be affected by the same architectural factors. Hence, one can study the identified groups, instead of the distinct metrics of interest. Moreover, since the few groups obtained are uncorrelated, they can be studied independently.

For each group of metrics, we form a single synthetic metric, which takes into account the positive or negative correlation of the group members. Depending on the relative importance of the metrics of interest, it is also possible to assign weights for the group members that compose the derived synthetic metric.

The proposed approach is applied on simulation data obtained by the ACID Sim Tools [1, 7], a tool suite developed by the authors for studying the runtime quality of transaction processing architectures.

Section II introduces the runtime design problems in transaction processing architectures and the case application scenario used to describe the proposed approach. Section III presents the analysis performed for discovering the correlation structure of the evaluated metrics. Section IV explains the procedure for constructing synthetic metrics and the paper concludes with a summary on the contribution of this work and a comment on its usability and the potential impact.

II. TRANSACTION PROCESSING ARCHITECTURES

Transactions provide system-wide Atomicity, Consistency, Isolation and Durability (ACID) guarantees for a set of operations that are initiated in a transactional server as a consequence of a transaction request.

Transactional architectures involve a series of complex design tradeoffs that make evident the need to investigate how the different architectural factors affect the metrics that determine the quality of service perceived by the end-user. Early detection of potential performance problems at the design phase is crucial, since the cost of correcting them after the software is implemented is disproportionally high. Typical runtime quality requirements for a successful architecture design include:

- High number of successfully processed (committed) transactions per time unit.

- Short response times, where response time is the time span between issuing the transaction request and the instant when the transaction either commits or aborts.
- High availability, where server availability is the ratio of time during which the server responds to transaction requests over the total time including server downtime, recovery and normal processing. While performing recovery from a failure, a server is unavailable to the clients. This implies that recovery times after failures should be minimized.

Recovery costs are determined by the size of the log files of the transactional server that in turn affects the I/O cost for the rollback of transactions that abort, as well as the use of locks in concurrency control. In essence, the size of the log file indirectly affects also the observed number of committed transactions per time unit.

With a quantitative evaluation technique, like the one proposed here, it is possible to quantify the significance and the relationships of the analyzed architecture design factors, when the system serves workloads with different degrees of distribution (localized to highly distributed transactions), different mixes of read-only and update transactions and different conditions of resource contention (I/O bound or CPU bound system configurations).

ACID Sim Tools is an integrated toolset and open source framework for simulating transaction processing architectures. It is based on a minimal set of assumptions that represent an object-based computational model (e.g. the OMG Core Object Model [8]) and is therefore appropriate for simulating transaction processing architectures like the ones that comply with the Enterprise Java Beans [4] and the OMG OTS standard [9].

A transaction invokes methods of objects that reside in one or more servers. These methods incur computations with specific CPU resource demands and are either read-only or alternatively update the state of the involved objects. For a given transaction request, the server receiving it is called coordinator and all other involved servers are called transaction workers.

Currently, we have implemented the basic two-phase commit (2PC) protocol for atomic commit processing of distributed transactions, along with two optimized variants, namely the 2PC presume commit and the 2PC presume abort protocols. For concurrency control, we have implemented the strict two-phase locking protocol, as well as the basic timestamp ordering protocol.

Our simulation framework provides metrics for evaluation and systematic experimentation on the tradeoffs between recovery costs and performance, when applying different combinations of protocols and protocol parameters. The provided metrics allow studying runtime quality by taking into account the degree of distribution and the degree of lock contention for the simulated workload. Apart from the generated measurements for the transaction workload as a whole, we have also implemented metrics for the different groups of transactions that exhibit a similar behavior, such as the group of all local transactions, the group of all distributed transactions and the different groups of transaction classes that compete for locks on the same objects.

The implemented metrics are:

- server availability
- the ratio of committed transactions (throughput)
- mean response time for all transaction groups
- mean blocking time for all distributed transactions, that represents the accumulated time-span between dispatching the workers' vote for the outcome of a transaction and the arrival of the coordinator's decision, during which processing in the worker is blocked for this particular transaction.

The proposed quantitative evaluation technique is demonstrated by a case application scenario represented by a synthetic transaction workload, which involves a number of transactional objects distributed among two servers. Table I introduces the considered object allocation and the associated object parameters as well as their method characteristics.

TABLE I. TRANSACTIONAL OBJECTS

Server	*Object*	*State Size (Kb) - exp-*[a]	*Method name*	*CPU demands (sec) - exp-*	*Read/ Write*
acp1	obj1	5	meth111	0.01	R
			meth112	0.05	R
	obj2	5	meth121	0.01	R-W
			meth122	0.01	R
			meth123	0.01	R
	obj3	5	meth131	0.04	R-W
			meth132	0.01	R
	obj4	5	meth141	0.01	R-W
			meth142	0.01	R
			meth143	0.01	R
	obj5	5	meth151	0.01	R-W
			meth152	0.01	R
			meth153	0.01	R
acp2	obj6	5	meth211	0.05	R-W
			meth212	0.05	R
	obj7	5	meth221	0.05	R-W
			meth222	0.01	R
	obj8	5	meth231	0.01	R
			meth232	0.01	R
	obj9	5	meth241	0.05	R-W
			meth242	0.05	R
	obj10	5	meth251	0.05	R-W
			meth252	0.01	R
	obj11	5	meth261	0.01	R
			meth262	0.01	R
	obj12	5	meth271	0.05	R-W
			meth272	0.05	R
	obj13	5	meth281	0.05	R-W
		5	meth282	0.01	R

a. exponential distribution

CPU time demands of the object methods obey the exponential distribution with the shown means. Table II displays the parameter values for the assumed execution environment. Since the aim of this evaluation is not to analyze the effects of the specific characteristics of the underlying execution environment, a minimal set of system-specific parameters is taken into account. These parameters are necessary for quantifying the evaluated runtime quality aspects for the studied architecture.

TABLE II. EXECUTION ENVIRONMENT CHARACTERISTICS

Network latency / message:		0.06 sec	
Server	***Disk Read Latency***	***Disk Wtite 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

We consider nine (9) transaction classes that are specified as sequences of method invocations representing the possible program paths in a transactional program. The transaction classes and their associated characteristics are shown in Table III.

TABLE III. TRANSACTION CLASSES

Transaction Class	***Methods invoked***	***Characteristics***
tr1	meth111, meth122, meth132	local (acp1) – read only
tr2	meth111, meth222, meth112	distributed – read/write
tr3	meth112, meth211, meth121	distributed – read/write
tr4	meth242, meth252, meth232	local (acp2) – read only
tr5	meth242, meth142, meth242	distributed – read only
tr6	meth242, meth141, meth251	distributed – read/write
tr7	meth262, meth272, meth282	local (acp2) – read only
tr8	meth262, meth152, meth262	distributed – read only
tr9	meth262, meth151, meth271	distributed – read/write

We studied the variation of the aforementioned metrics, in terms of the architectural factors and workload parameters (MIT and MITofSF) shown in Table IV. A checkpoint is a periodically invoked log truncation, which removes log entries that are no longer needed for server recovery from a failure. Frequent checkpoints result in faster recovery, but a substantial amount of additional I/O workload is imposed during normal operation. Multiprogramming level (MPL) represents the maximum number of transactions that are allowed to run concurrently in a server. Concurrency control in the experiment adheres to the widely used strict two-phase locking (2PL) scheme. To avoid deadlocks, the servers maintain a timer that is activated upon initiation of each transaction and when the timer expires the transaction is aborted. Transaction timeout (TT) is the time-span between the initiation of the transaction processing and the expiration of this timer.

TABLE IV. FACTORS THAT INFLUENCE RECOVERY COSTS & PERFORMANCE

Factors	***Level 1***	***Level 2***	***Level 3***
Atomic commit protocol (ACP) - all servers -	2PC PRN Presume Nothing	2PC PRC Presume Commit	2PC PRA Presume Abort
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.[a] - all transaction classes -	0.6	0.4	
Mean interarrival time of server fail -stop failures (MITofSF) – exp.[a]	18 m	5 h 51 m	12 h

a. exponential distribution

The full experiment (all possible combinations of factor levels) includes 486 simulation runs, where the simulated time for each run was 55h 30m. Sufficiently long simulation runs ensure that the metrics are studied when being in steady-state. Alternative experiment designs with fewer simulation runs are the uniform designs that we used in [5].

III. DISCOVERING GROUPS OF CORRELATED METRICS

In the described architecture design, the metrics of interest are influenced by a tradeoff between recovery cost and performance. However, the problem is complicated due to the fact that there is no precise characterization of the recovery cost, since it is not directly measurable and the fact that performance is quantified by a large number of metrics (shown in Table V), where each metric discloses a different aspect of the runtime behavior. In addition, the availability of the two servers (*availability0, availability1*) is quantified, under the considered levels of fault load.

TABLE V. RUNTIME QUALITY METRICS FOR THE CONSIDERED WORKLOAD

	Throughput	***Mean response time***	***Mean blocking time***
All distributed transactions	tput_distr	response_distr	blocking_distr
All local transactions	tput_local	response_local	
Group 1: tr1, tr3	tput_confl0	response_grp0	blocking_grp0
Group 2: tr4, tr5, tr6	tput_confl1	response_grp1	blocking_grp1
Group 3: tr7, tr8, tr9	tput_confl2	response_grp2	blocking_grp2

Thus, the analysis, which was conducted using the SPSS 16 statistical software, had to take into account the measured variations in 14 quality metrics of the runtime performance, plus two metrics that quantify the servers' availability. This is a problem with high dimensionality and at least one latent quality feature, i.e. the recovery cost.

The studied metrics depend on architectural factors that exhibit non trivial interdependences, which either cannot be quantified directly, like the recovery cost, or they are unknown and should be discovered and subsequently explained. Dependence of two or more metrics on the same architectural factors induces a correlation between them. The examination of the correlation structure of the evaluated metrics reveals various statistically significant correlations either positive or negative.

The reduction of problem dimensionallity is achieved by computing new, fewer variables that contain most of the information present in the original ones. The idea is to create groups of highly correlated dependent variables representing the quantified metrics.

The appropriate procedure to follow is Factor Analysis (FA), a methodology that discovers new variables (called factors) in the original dataset, by exploiting their correlation structure. To avoid confusion with the term "factor", as it is also used for the architectural factors of Table IV, the new variables are called from now on "components", a terminology consistent with the applied statistical analysis. Each component represents a group of correlated metrics and this grouping in effect reduces the dimensionality of the original data. Furthermore, the new variables resulting from such a procedure are uncorrelated and can be studied separately with respect to the considered architectural factors.

Factor analysis requires that the dependent variables are normally distributed and for this reason we applied a transformation based on the rankings of the values in each metric. We used Blom's transformation [3] that utilizes the ranks r_i of the values and the cumulative Normal distribution function $\Phi^{-1}(.)$. The formula used in this transformation is

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

It is important to note that the applied transformation preserves the correlation structure of the original dataset. For example, in Fig. 1 a strong negative correlation between *tput_local* and *blocking_distr* is evident in the original and in the transformed variables. Moreover, we observe that the normalization (bottom panel) portrays better their correlation.

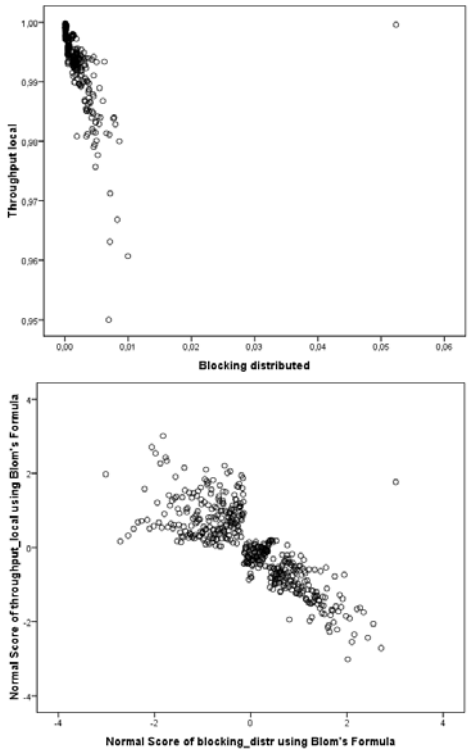

Figure 1. Strong negative correlation between *tput_local* and *blocking_distr* in the original and transformed dataset.

On the transformed normalized dependent variables (metrics) we performed a FA procedure based on *Principle Components with Varimax rotation*. The new variables, i.e. the component scores, were calculated by the Anderson-Rubin Method ensuring that the variables produced have a mean of 0, a standard deviation of 1, and are uncorrelated.

The FA resulted in 3 components that explain 88.81% of the variance of the 16 original variables. This means that we can have a significant reduction of the output space in dimensions by exploiting the correlation structure of the normally transformed outcomes.

Table VI shows the loadings of the variables on the three components and therefore the grouping of the studied metrics. The sign shows the direction of the detected correlation. For example, *tput_local* is loaded to the same component as *blocking_distr* but their loadings have different signs due to their negative correlation. This finding is explained by the following metric dependency: longer blocking times for distributed transactions result in lower throughput for local transactions. Blocked transactions can not release the locks that have been acquired until the decision of the coordinator arrives, hence, local transactions are queued for one or more locks, with a significant probability to abort, due to a transaction timeout. Also, when the servers' availability is low, longer blocking times are obtained, as a consequence of the caused delays in communicating the messages that control the transaction execution.

TABLE VI. ROTATED COMPONENT MATRIX OBTAINED BY PRINCIPLE COMPONENTS ANALYSIS WITH VARIMAX ROTATION

Normal Score[a] using Blom's Formula	Component		
	1	*2*	*3*
availability0	**-0.800**	0.009	0.115
availability1	**-0.794**	-0.002	0.106
throughput_local	**-0.831**	0.132	0.406
throughput_distributed	-0.131	-0.125	**0.978**
throughput_confl0	-0.166	0.098	**0.955**
throughput_confl1	-0.138	-0.227	**0.950**
throughput_confl2	-0.140	-0.239	**0.947**
response_local	0.031	**0.724**	-0.389
response_distr	0.095	**0.953**	-0.203
response0	-0.036	**0.948**	-0.205
response1	-0.017	**0.975**	0.122
response2	-0.013	**0.989**	0.040
blocking_distr	**0.967**	0.062	-0.054
blocking_0	**0.935**	0.068	-0.126
blocking_1	**0.944**	0.007	-0.034
blocking_2	**0.948**	0.038	-0.050

a. Extracted Method: Principal Component Analysis
Rotation Method: Varimax with Kaiser Normalization

Table VII shows the groups found by the performed statistical analysis. High values for component 1 (C_1) are related to high values of blocking times for distributed transactions and low values of servers' availability and throughput for local transactions and vice versa. Component C_2 groups all measured response times and C_3 groups the throughput of distributed transactions and the different groups of lock-conflicted transactions that are positively correlated with their variables.

TABLE VII. GROUPS OF CORRELATED QUALITY METRICS

	Total variance explained (%)	***Metrics***
Component 1	35.35	*availability0* (-), *availability1* (-), *tput_local* (-), *blocking_distr* (+), *blocking_grp0* (+), *blocking_grp1* (+), *blocking_grp2* (+)
Component 2	27.64	*response_local* (+), *response_distr* (+), *response_grp0* (+), *response_grp1* (+), *response_grp2* (+)
Component 3	25.83	*tput_distributed* (+), *tput_confl0* (+), *tput_confl1* (+), *tput_confl2* (+)

IV. CONSTRUCTION OF SYNTHETIC METRICS

The groups of strongly correlated metrics found by Factor Analysis are:

- component C_1, a new variable that is a computed "score", for the variability of all metrics, mostly affected by I/O, due to checkpoints and recovery
- component C_2 summarizes the variability of all response time metrics and
- component C_3 summarizes the variability of throughput metrics, that are mostly affected by communication latencies.

An outstanding problem with the computed values for the mentioned components is that it is not possible to intuitively interpret them, in order to assess the impact of the metric groups on the overall system's behavior.

For this reason, when applying similar statistical analyses it is often desirable to construct artificial metrics that express the quantified variability in an intuitive way, in order to be able to interpret the metric values.

An appropriate synthetic metric for each component should take into account the relative importance of each group member, with respect to the quality design goals. Besides the possibility of using relative weights for the metrics of a component, the correlation type (positive or negative) enforces the relation of the new synthetic metric in terms of the member metrics.

Let us consider component C_1 in Table VII. Since the goal is to increase throughput and the servers' availability, these metrics should be combined in a way such as increase of throughput or availability increases the synthetic metric value. On the other hand, because the measured blocking times are inversely correlated to availability and throughput, increase of blocking time should result in decrease of the synthetic metric value.

Although we are often interested in studying the runtime behavior with respect to part of the workload that concerns a particular characteristic (e.g. distribution or resource sharing), with a synthetic metric we summarize information for the entire workload. Therefore, we don't use metrics that represent information already contained in other metrics. In the synthetic metric constructed for component C_2, we take into account *response_local* and *response_distr*, but we ignore (by setting weight 0) *response_grp0*, *response_grp1* and *response_grp2* that are already reflected in the two utilized metrics.

The synthetic metric for component 1 of our experiment is computed as follows:

$$component1 = \frac{tput\_local * (\frac{availability0 + availability1}{2})}{blocking\_distr}$$

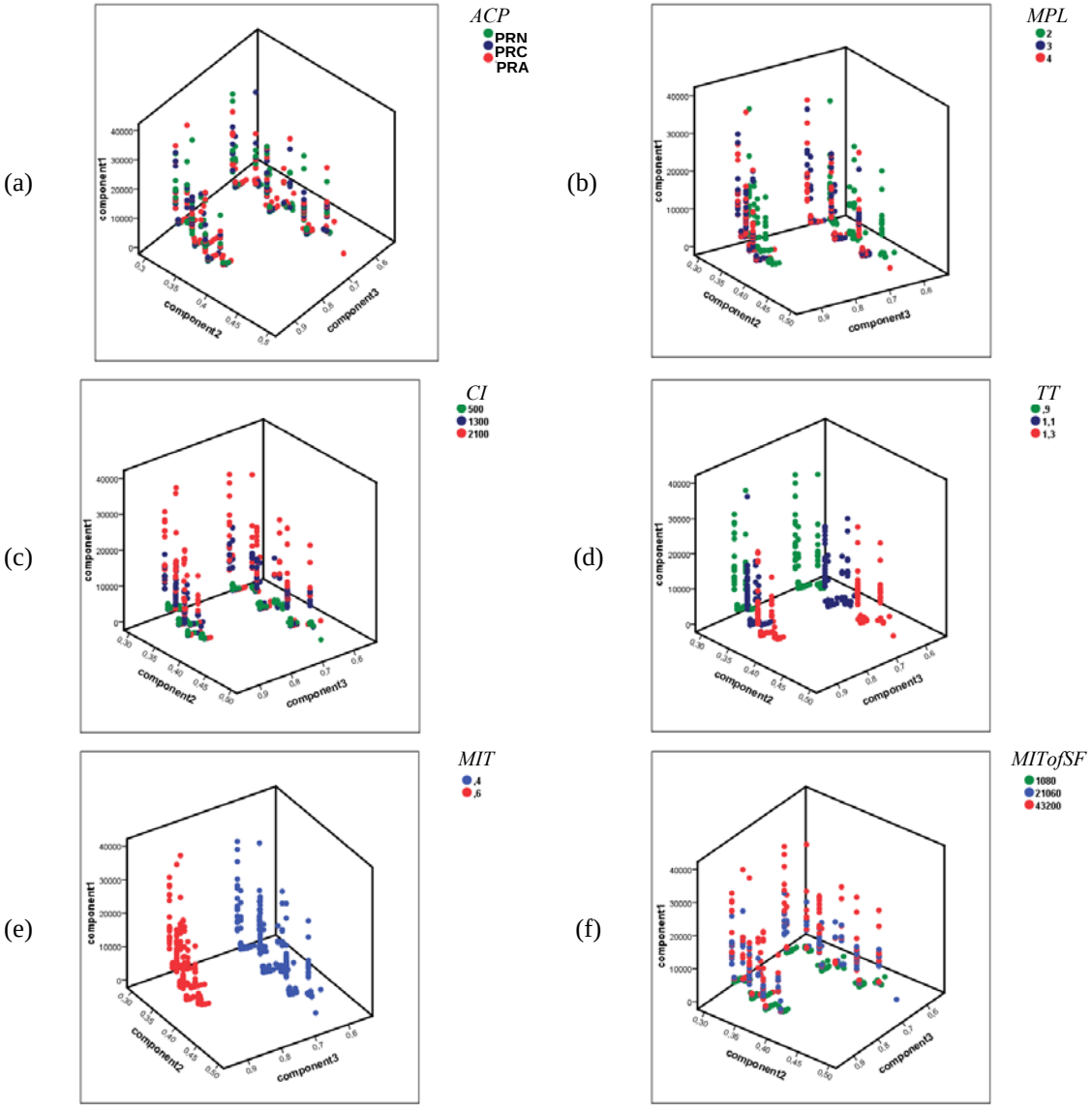

Figure 2. 3-d scatterplots for the three synthetic performance metrics with respect to the architectural factors.

Since all simulated atomic commit protocols are blocking protocols, the obtained blocking times are always greater than 0, hence the proposed metric does not become undefined. High values for *component1* indicate improved performance, as a consequence of the way the utilized metrics were combined.

The synthetic metric for *component2* is given by:

$$component2 = \frac{response\_local + response\_distr}{2}$$

In contrast to the previous synthetic metric this one indicates improved performance for comparatively smaller values, which in fact show shorter response times. The range of *component2* includes values from 0 to infinity, where infinite time indicates transactions for which there is no decision made by the coordinated servers. However, due to the applied transaction timeout strategy, this possibility is never realized.

The synthetic metric for *component3* is calculated by:

$$component3 = \frac{tput\_confl0 + tput\_confl1 + tput\_confl2}{3}$$

The range of *component3* includes values from 0 to 1, where 0 indicates that there are no committed transactions and 1 that there are no aborted transactions.

Fig. 2 shows the 3-d scatterplots obtained for the three synthetic metrics, with respect to the six architectural factors of Table IV.

The scatterplots show that the 3-d points with coordinates the values of the three synthetic metrics are clustered with respect to the architectural factors. First of all, there is a clear grouping for the two levels of the *MIT* (Fig. 2e). The groups are clearly distinguished and are almost parallel and this pattern is preserved in all plots. Low throughput (*component3*) corresponds to mean interarrival time 0.4 sec, meaning that arrivals of transaction requests that are more frequent than 0.6 sec in essence deteriorate the measured throughput. The reason is that too many transactions that run concurrently result in increased contention for the shared resources (CPU, I/O bandwidth and the available locks) and a significant number of them are eventually aborted. We also observe that when the mean interarrival time is 0.6 sec, as a consequence of lower resource contention, response times are improved.

Another clear clustering is apparent with respect to transaction timeout (Fig. 2d). Short response times (*component2*) correspond to TT=0.9 (green points) and larger values of transaction timeouts result in increasingly longer response times. This is explained by the fact that higher timeout values result in a comparatively higher number of committed transactions that affect the mean response time, but these additional transactions have in general response times that are close to the timeout value.

High values of *component1* (red points) correspond to checkpoint intervals 2100 sec, but in this case recovery times are long and perhaps a better design option is to use checkpoint intervals with period 1300 sec (blue points). Fig. 2f confirms the negative effect of frequent server failures (green points) on the measured availability and blocking times (*component1*).

Finally, Fig. 2b clearly shows that when using MPL=2 (green points), response times are deteriorated, due to the incurred queuing delays for acquiring access to the CPU. A less risky option is the use of three threads (blue points), because an even higher number of threads results in severe lock contention and higher response times.

V. Conclusion

This work focuses on evaluating the impact of design decisions on runtime quality aspects of software architectures. While it is desirable to use a wide range of metrics that provide the opportunity to study different aspects of the runtime behavior, these metrics are usually characterized by complex correlations that hinder the perception of the overall system behavior.

With regard to this problem we developed an approach that first detects the existing correlations between the analyzed metrics and groups them in uncorrelated components. This opens prospects to construct new synthesized quality metrics summarizing the overall system performance in a concise and easy to interpret manner.

The proposed approach was demonstrated by a case application scenario concerning a transaction processing architecture. The obtained results are presented in 3-d scatterplots that allow assessing the impact of the architectural factors on the system behavior.

The most prominent prospect of our anlysis is the possibility to be applied on performance tuning of architecture designs that may be studied under workloads represented by benchmarking applications. A future research perspective is the extension of this work, in order to exploit the obtained results in a systematic multi-objective optimization of the system's performance.

References

[1] ACID Sim Tools Site, http://mathind.csd.auth.gr/acid/html/index.html (last access: 27th of March 2009)

[2] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, 2nd ed., SEI Series in Software Engineering, Addison-Wesley, 2003.

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

[4] B. Burke and R. Monson-Haefel, Enterprise Java Beans 3.0, O'Reilly, 2006.

[5] P. Katsaros, L. Angelis and C. Lazos, "Performance and effectiveness trade-off for checkpointing in fault-tolerant distributed systems", Concurrency and Computation: Practice and Experience, vol. 19, 2007, pp. 37-63, doi: 10.1002/cpe.1059

[6] Liu, Y., Zhu, L. and Gorton, I., "Performance assessment for e-Government services: An experience report", Proc. 10th International Symposium on Component Based Software Engineering (CBSE), Springer LNCS 4608, 2007, pp. 74-89

[7] A. Mentis, P. Katsaros and L. Angelis, "ACID Sim Tools: A simulation framework for distributed transaction processing architectures", Proc. 1st Int. Conf. on Simulation Tools and Techniques (SimulationWorks Industry Track), ICST, 2008, http://eudl.eu/?eudlQuery=SimulationWorks%202008

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

[9] Object Management Group, Transaction Service Specification, version 1.3, OMG Technical Committee Document ptc/2003-03-08, March 2003