Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 261 (2010) 109–130
www.elsevier.com/locate/entcs

A Massively Scalable Architecture For Instant Messaging & Presence

Jorrit Schippers [1] Anne Remke [2]

Design and Analysis of Communication Systems, University of Twente, Enschede, The Netherlands

Henk Punt

Hyves, Amsterdam, The Netherlands

Maarten Wegdam

Novay, Enschede, The NetherlandsA

Boudewijn Haverkort

Design and Analysis of Communication Systems, University of Twente, Enschede, The Netherlands, and Embedded Systems Institute, Eindhoven, The Netherlands

Abstract

This paper analyzes the scalability of Instant Messaging & Presence (IM&P) architectures. We take a queueing-based modelling and analysis approach to find the bottlenecks of the current IM&P architecture at the Dutch social network Hyves, as well as to analyse three alternative architectures: *evolutionary partitioning*, *aggregated*, *batched presence updates* and *presence subscriptions*. We use the Hierarchical Evaluation Tool (HIT) to create and analyse models analytically. Based on these results, we recommend a new architecture that provides better scalability than the current one.

Keywords: Instant Messaging and Presence Architecture, Queueing Models, Scalability

1 Introduction

The popularity of an online service depends on many factors, but no matter how many people are willing to use a service, it will not be successful when the architecture is not able to handle the load. An example of a service that had to spend a

[1] Email:j.k.schippers@alumnus.utwente.nl
[2] Email:anne@cs.utwente.nl

doi:10.1016/j.entcs.2010.01.008

lot of effort improving the architecture after it became popular is Twitter [11]. An instant messaging & presence (IM&P) service allows users to exchange messages and distribute presence information. Whereas messages are exchanged between one user and another, updates to presence information are multicasted from the source to all related users (contacts). An IM&P architecture needs to forward messages and exchange presences in a timely manner, as users expect instanteneous response, especially for messages. In addition to that, the architecture must provide the user with the presence information of all contacts when the user connects to the service. Hyves is a Dutch social network and offers all kinds of social applications, such as personal profiles, weblogs and IM&P. The current IM&P architecture is sufficient for the current workload, but Hyves engineers expect that it will not be able to handle a tenfold increase.

The main goal of scalability [5], [4] analysis is to compare an architecture under different workloads. For this it is not necessary to know each performance aspect in detail, as constant aspects do not contribute to comparisons. In the following we call an architecture scalable, if changing the amount of hardware resources allows the system to handle a proportionally changing workload while the performance remains the same.

The contribution of this paper is that we evaluate and compare different architectures with respect to their scalability and their applicability for an IM&P service, with the Hyves architecture as use-case. The three alternative architectures are *evolutionary partitioning*, *aggregated, batched presence updates* and *presence subscriptions*. We take a queueing network-based modelling and analysis approach to find the bottlenecks of architectures. The Hierarchical Evaluation Tool (HIT) is used to create and analyse IM&P architectures. HIT translates these models then to open queueing networks and can be used to compute, e.g., throughput and response times.

Although several high-level descriptions of IM&P architectures are available on the internet, we are not aware of previous work on their analysis. However, a number of studies of with other purposes have been published, that mainly develop an analytically solveable model for an existing computer or software architecture. In [13] a simple webserver architecture is modelled and analysed using queueing networks. A comparison between two Enterprise Java Bean architectures using queueing networks is performed in [8].

This paper is structured as follows: we detail our modelling and analysis approach in Section 2. In Section 3 we describe the current IM&P architecture and its bottlenecks. Section 4 contains detailed descriptions of three alternative architectures. In Section 5 we describe the analysis results for the current and alternative architectures. Section 6 contains the conclusion and recommendations for future work.

2 Modelling and analysis approach

We use the Hierarchical Evaluation Tool (HIT) to create and analyse models. These models are constructed using a graphical notation in the supporting tool HIT-GRAPHIC [2,3]. HIT translates the models into an open queueing network and uses the DOQ4 solver to calculate measures, such as utilisation, throughput and response time. The term scalability relates to both the workload and the resources of a system. Hence, a scalability model captures the relation between workload and resources. We model both, the workload and the resources using HIT. Using HIT, an architecture can be represented by a single model, while the individual parts can each be modelled at convenient abstraction levels. At the upper level, parameters can be tweaked to model different user characteristics. This can be done to model a growth of the number of users or to model the change of behaviour of the average user. At the lower levels of the model, the utilisation of the resources is measured. In our models, we consider the following resources:

Database access

Our experiments have shown that the raw database performance is much higher than the performance of the database access layer in the server software. Building the database request and interpreting the database response on the client side takes more time than retrieving the data from memory. This is related to the fact that databases in the current Instant Messaging & Presence system are optimised such that every read or write access utilises indices. The table containing all presence information is loaded entirely into main memory, further lowering the processing time in this component.

This is modelled by ignoring the pure database access time. Real-world phenomena such as lock contention, replication delay and query caching are ignored as they only affect the performance of a single database, not the scalability of an architecture. Database access is modelled as a $M|M|\infty|\infty|\infty|PS$ queue, where read and write requests have different service demands. This results in a low delay that is independent of the number of processes accessing the database for low amounts of simultaneous queries, but rapidly increases when the load approaches some threshold. In this way, the delay is insignificant for a normal load, but will be prohibitive when the component is overloaded, indicating a lack of scalability. Our experiments indicate that write requests are slightly faster than read requests. The reasons for this difference is that the database is stored in main memory and that read requests require more processing in the Python database abstraction layer, which is used in the current architecture. The databases in the model have a service rate of 100,000 service units per second, while a write request requires 22 service units and a read request 28 service units. Besides the processing time, the model also provides measures such as query rate. These measures are used to determine the relation between workload and component behaviour.

Network links

The diagram of the current architecture (Figure 1) shows that the database master node is associated with many slave nodes. Incoming updates to the master database are translated in replication traffic between the master node and its associated slaves. This replication traffic is simply a duplicate of the incoming update statements: each statement is sent to each slave. At peak times, this can cause a high load on the outgoing network link, hence the links themselves are modelled and measured. Using event-based networking libraries such as *libevent*, the number of connections per link is virtually unlimited, as long as the aggregate traffic per link is bound. Therefore we only model the incoming and outgoing traffic per network link. Each network link is modelled as a $M|M|\infty|\infty|\infty|PS$ queue with a service rate of one gigabit per second in each direction.

Each model is analysed to find the relation between workload and utilisation of the network and database resources. In a series of runs, the workload and number of resources are increased linearly. The change in throughput of each resource instance indicates if the resource is scalable or not. Ideally, each network link sees an equal throughput if the workload and number of network are increased proportionally. The measurement then shows a constant throughput per network link. If the measurement shows increasing throughputs, each resource is utilised more, despite the fact that the ratio of workload and resources is kept constant.

The workload is a combination of the current peak rates for the five actions users can perform on the service: log in, log out, send a message, receive a notification and change presence information. The peak rates are derived from measurements on the current architecture. These five rates are used together as one unit of workload in all architectures. The Facebook chat service [7] shows that response time is not an issue for presence propagation. Instead, the delay for transferring instant messages must be low, as users might become dissatisfied with the service if the messages do not arrive immediately. Hence, we will analyse the response time of the message send action for each architecture. In addition to that, we compare the architectures by analysing the relation between workload and number of machines necessary to handle that workload. We assume that each architecture is able to handle the current peak workload with the current number of machines, which is 37. We measure the utilisation of the resources of each architecture at that point. For every increase of the workload, we determine the number of machines necessary to keep the utilisation at the initial level.

3 Case study: current architecture

The current IM&P architecture is shown schematically in Figure 1. *Application nodes* and *slave nodes* are related: physical machines contain both the application node and the database slave node to decrease look up times. The dotted borders of application nodes, slave nodes and clients indicate that the number of these nodes can be changed. The grey star in the centre of the diagram visualises the connections between all application nodes. Application nodes determine the behaviour of the

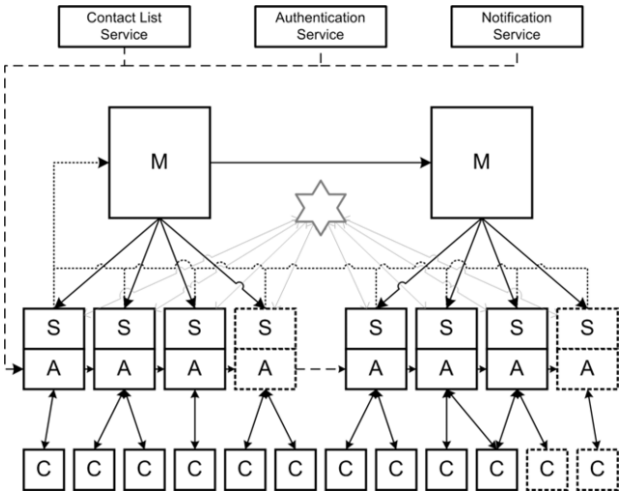


Fig. 1. Overview of the current architecture

system. They handle connections from clients, interpret requests, call external services or other nodes to fullfill these requests and create the response to be sent to the clients. Clients connect to one of the application nodes as determined by a weighted round robin scheme. The weight is manually set by the system operators to represent the relative capacity of each machine. Currently, all machines are equal and hence have equal weights. Messages and presence updates are submitted to this application node. Instant messages are sent from the originating application node to the application node to which the destination is connected. The mapping between users and application nodes is stored in the database. For every arriving instant message, the application node looks up this mapping to determine where the message should be routed to. For this look-up, no additional network traffic is required, as it is done on the local database slave node. Presence updates are propagated similarly. When a user changes his presence information, the update is sent directly to the application nodes to which the contacts are connected. The difference between presence forwarding and message forwarding is that presence information is stored persistently. To achieve this, the application node forwards presence updates to the database *master node*. This node is connected to all slave nodes and replicates updates to these slave nodes.

The protocols used in this architecture are the MySQL protocol for data replication between master and slaves and for the traffic between the slave nodes and the application nodes and the Extensible Messaging & Presence Protocol (XMPP) for the client-application traffic. The MySQL protocol is unicast, such that adding a slave causes additional outgoing traffic on the master node. The current architecture does not enforce a particular network layout. In practice, the machines are dispersed over various data centres and locations within those data centres. The database of the current architecture stores user, presence and session information for each user that has ever connected to the service. The user information that is stored is just the username and the user identification number (userid). Other information is stored in external services. The presence information consists of a status

and a free text field. The status field contains one of the six statuses that users can set while they are online: online, busy, be right back, away, on the phone and out for lunch. The free text field contains the entire protocol message of a presence update, as the XMPP standard requires implementations to maintain this information even when users are offline. The session information includes an identification number of the application node to which a user is connected and the identification number of the connection within that application node. Empty values for these fields indicate that the user is not connected.

A known bottleneck of the master-slave database architecture is the master, as it receives 100% of the write load of the system and is responsible for replicating the updates to all slaves. The outgoing network link is a bottleneck, as the number of packets is proportional to the number of slaves. Increasing the number of slaves does not decrease the load of replicated update queries per slave, as each slave needs to process all updates. With an increasing write load, the resources left at the slave nodes to serve read requests become smaller and smaller, thus increasing the response time of the system [9]. This means that the current architecture is not scalable: the architecture is not able to increase its capacity linearly by adding more hardware.

Based on the approach outlined in Section 2 a model was developed using the graphical modelling tool HITGRAPHIC and the associated textual modelling language HI-SLANG. The architecture layer of the model is displayed graphically in

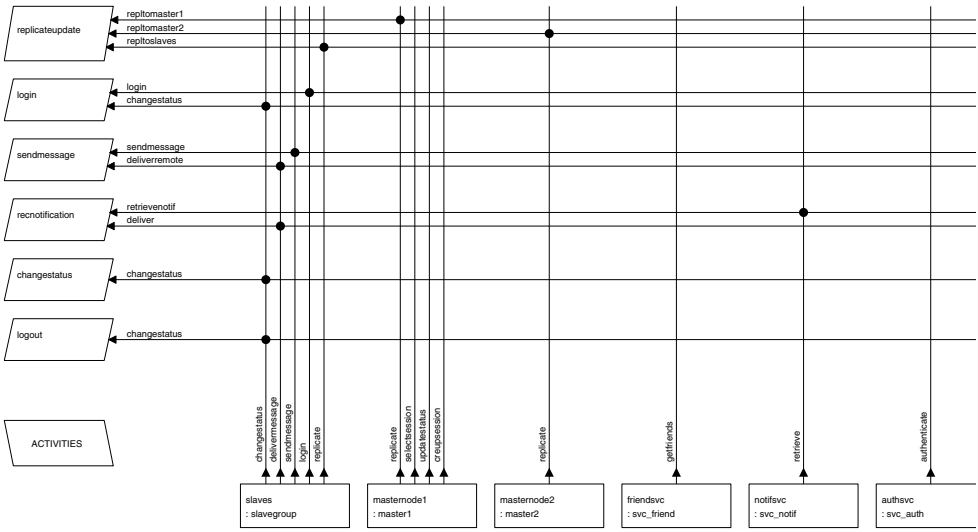


Fig. 2. Top layer of the HIT model of the current architecture

Figure 2. The services of the architecture are displayed as parallelograms on the left side of the model. The model contains one service for each action. The bottom row of rectangles represent the underlying components of the architecture. The first three rectangles refer to the internal architecture components: slaves, master node 1 and master node 2. The remaining components refer to the external services. The

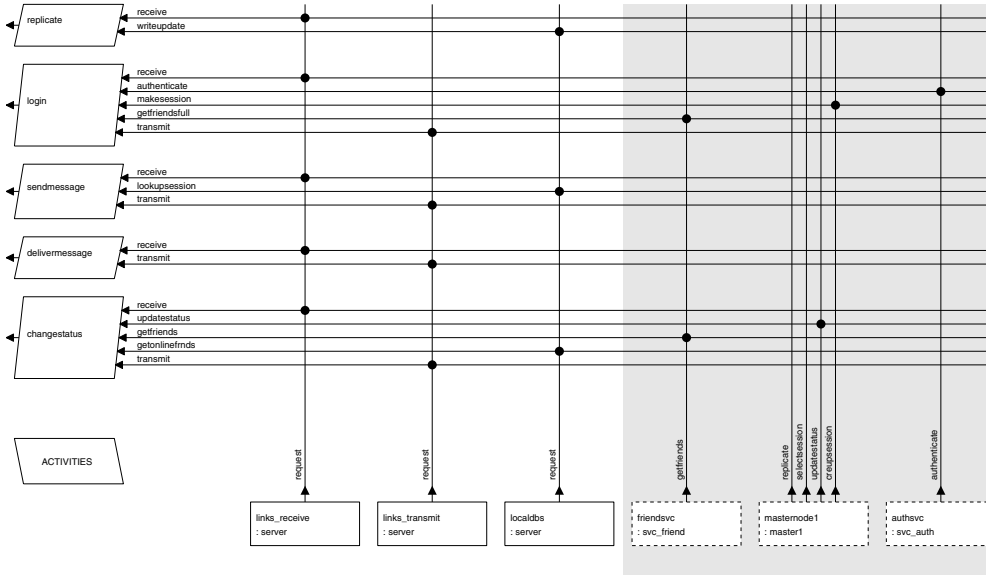


Fig. 3. Hit model of slave nodes in the current architecture

services of the architecture invoke services offered by the underlying components. Each used service is represented by a horizontal line originating at the service that uses an underlying service, such as *login*. Each offered service is represented by a vertical line originating at the corresponding component. The dot on the intersection indicates the relation between the used service and the offered service. The model shows that that the services *login*, *changestatus* and *logout* are similar, as they all use the service *changestatus* of the *slavegroup* component.

The *replicateupdate* service is not directly associated with an action. This service represents the asynchronous replication of presence updates from master nodes to slave nodes. In this way, replication delays do not lead to delays in the *login*, *changestatus* or *logout* processes. Because of modelling constraints, the group of all slaves is modelled as one component. Arrays of identical components are possible in HIT, but utilising this functionality would require solving the model using simulation instead of analytical methods. The implementation of the services of the slave group takes into account the number of slaves represented by the group. The HI-SLANG code executes a certain operation n times when it needs to execute that operation on all n slaves. Figure 3 shows the HITGRAPHIC model of the *slavegroup* component. Each of the high level services of the slave node is mapped to services of components. The most important components are the links and the local databases, as they determine the scalability properties. Components with a grey background are shared instances with other parts of the model, such that invocations on these components from anywhere in the model contribute to the same measurements.

The model parameters are listed in Table 1. Table 2 lists values for these parameters, as they have been found using various methods: the rates of the actions

Workload parameters	
loginrate	Rate of login actions (actions per second)
logoutrate	Rate of logout actions (actions per second)
sendmessagerate	Rate of message sending (actions per second)
notificationrate	Rate of notifications (actions per second)
statuschangerate	Rate of status changes (actions per second)
numslaves	Number of slave nodes
Architecture parameters	
avgrostersize	Average number of persons in a contact list
onlinepct	Percentage of users online at peak (bytes)
avgmessagesize	Average size of an instant message (bytes)
avgstatusc2ssize	Average size of a status message, client to server (bytes)
avgstatuss2csize	Average size of a status message, server to client (bytes)
avgquerysize	Average size of a replicated write query (bytes)
avgloginc2ssize	Average size of login conversation, client to server (bytes)
avglogins2csize	Average size of login conversation, server to client (bytes)
readqueryload	Relative load of a read query on database (workload units)
writequeryload	Relative load of a write query on database (workload units)
linkspeed	Speed of an unidirectional network link (bits per second)
dbspeed	Speed of a database (workload units per second)

Table 1
Model parameters for the current architecture and workload

are the top values of measurements on the current architecture. The packet sizes have been deduced from network traces on this architecture. The database load estimates were estimated using performance tests. A more detailed description of the parametrisation of the model is given in [12].

4 Architecture proposals

In the following we analyse the scalability of three different architectures, *evolutionary partitioning* is discussed in Section 4.1, *aggregated, batched presence updates* is analysed in Section 4.2 and *presence subscriptions* is evaluated in Section 4.3.

4.1 Architecture 1: Evolutionary partitioning

By partitioning the master and slave nodes increased load can be handled [1]. Figure 4 shows the new composition of the components in this architecture. The architecture contains *application nodes* that have the same role as in the current

Workload parameters		Architecture parameters	
loginrate	210	avgrostersize	140
logoutrate	200	onlinepct	0.06
sendmessagerate	200	avgmessagesize	200
notificationrate	240	avgstatusc2ssize	150
statuschangerate	10	avgstatuss2csize	230
numslaves	35	avgquerysize	450
		avgloginc2ssize	1150
		avglogins2csize	13,000
		readqueryload	28
		writequeryload	22
		linkspeed	1,073,741,824
		dbspeed	100,000

Table 2
Parameters used in the current architecture

architecture. However, in this architecture the database *slave nodes* are separate machines that are grouped in one or more *partitions*. Each partition consists of one database *master* and one or more database slave nodes. Each user is assigned

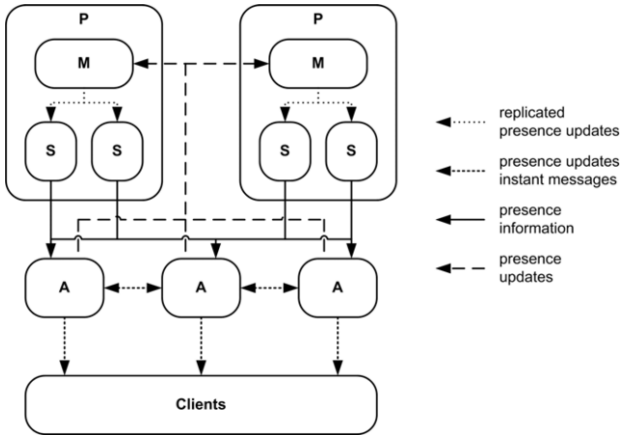


Fig. 4. Overview of the evolutionary partitioning architecture

to a partition by a partitioning algorithm such as Consistent Hashing [6]. This means that the presence information is divided over many partitions. When a user logs in, the application node needs to look up presence information of the users in the contact list on all partitions. Using asynchronous programming, queries to all partition can be executed in parallel, reducing latency. Users can connect to any application node available. Upon reception of a presence update from a client, it submits the update to the master node of the users' partition, which replicates it to the slaves of that partition. Instant messages are transmitted directly between

application nodes, similar to the current architecture.

4.2 Architecture 2: Aggregated, batched presence updates

The Facebook chat architecture provides Instant Messaging and Presence to nearly 200 million users [7], [10]. Instead of distributing presence updates instantly to other users, this architecture batches presence updates and sends them at fixed intervals. The partitioning approach is just used for exchanging messages and to keep all presence information on centrealised presence nodes. The same approach is used in the following to define an architecture that is suitable for Hyves. Figure 5 gives

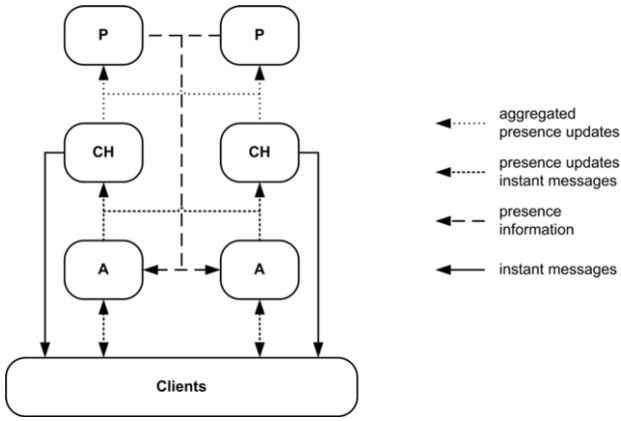


Fig. 5. Overview of the aggregated and batched presences approach

an overview of the alternative architecture. Each channel node forwards messages and stores presence information for a portion of the user base. Clients connect to one of the *application nodes* and the *channel node* that stores their messages. Application nodes forward incoming messages to the channel node of the receiving user. Incoming presence updates are forwarded to the channel node of the user that sent the presence update. They are stored at channel nodes and transmitted in batches at a given interval to all *presence nodes*, which store presence information for all users. Application nodes poll presence information for their connected users at a given interval. They fetch presence information for all users from the contact lists and forward this information to their clients.

Channel nodes are the authoritative source for presence information, while application nodes do not store presence information at all. When an application node tries to send a message to a channel node for a user that is unavailable, the channel node generates an error, which is transported back to the client. The capacity requirements on their links force us to reduce the amount of presence information kept per user. At this moment, a user can select one of six possible presence statusses as well as store information in a 500 byte free text field. Only the presence status can be stored in this architecture. We store this information using three bits of data.

4.3 Architecture 3: Presence subscriptions

Presence subscriptions are used in the Windows Live Messenger architecture [14]. The difference is that application nodes subscribe to presence updates at database nodes. This reduces the amount of network traffic. Figure 6 gives an overview of

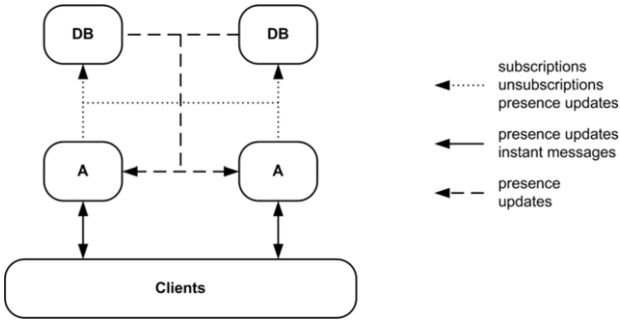


Fig. 6. Overview of the subscription-based architecture

the subscription-based architecture. Users can connect to any of the *application nodes*. Each *database node* stores subscriptions and presence information for a portion of the user base. Each database node and application is a single machine. There are no master or slave nodes in this architecture. When a user logs in, the application node sends a subscription to each database node that stores information for a user in the contact list. Whenever one of those users updates their presence, the database node uses the list of subscriptions to forward the presence update to the application nodes. In this way, presence is only forwarded to nodes that actually need this information. Application nodes cache presence information for their users' contacts. The subscription mechanism ensures that this cache always contains up to date information. Instant messages are forwarded directly between application nodes, similar to the current architecture. While in that architecture and in the first proposal each instant message meant a lookup in the presence database, in this architecture application nodes can use their internal cache. Presence subscriptions are removed when a user logs out.

5 Analysis results

We first analyse the bottlenecks in the current architecture. Figure 7 shows the results of this experiment on the master and slave databases. The x-axis contains the factor by which the workload is multiplied, which is equal to the factor by which the number of slaves is multiplied. The starting values are one time the unit of workload and 35 slave nodes, as that is the current amount of slave nodes. The y-axis shows the number of queries per second for the first master database and one slave database. Two observations can be made: first, both numbers increase linearly with the increased load, despite the fact that more slaves have been added. Secondly, the two curves are quite close to each other, with the slave database slightly more loaded than the master database. This relates to the fact that the slave database handles both the replicated update queries arriving from the master

and the read queries executed by the application. The measurements end after 4.5 workload units, as the queues become overloaded at that point.

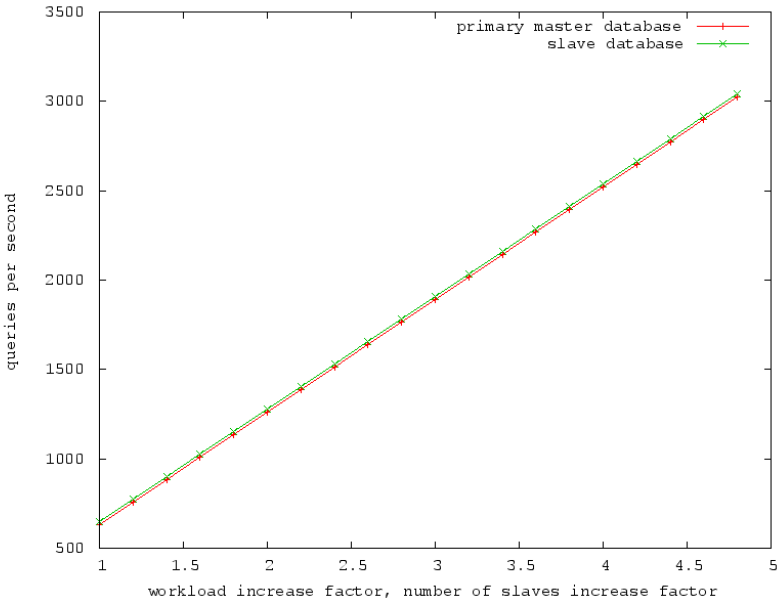
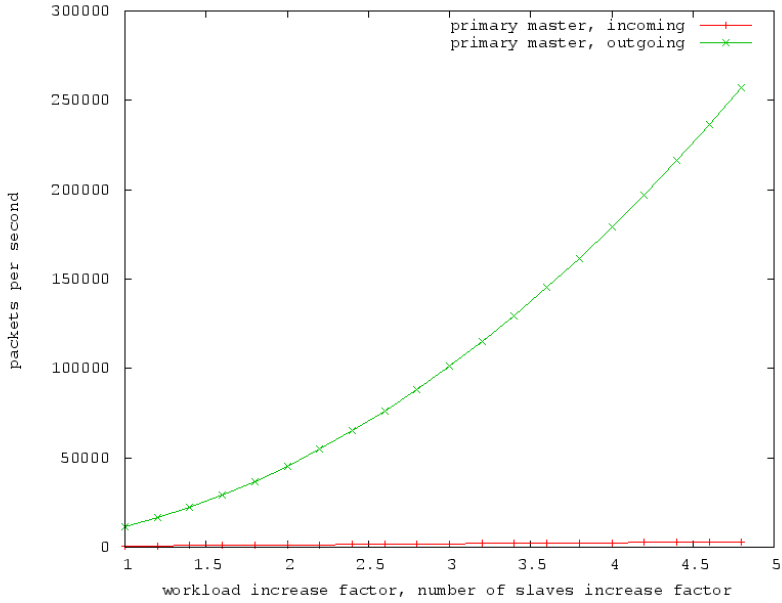


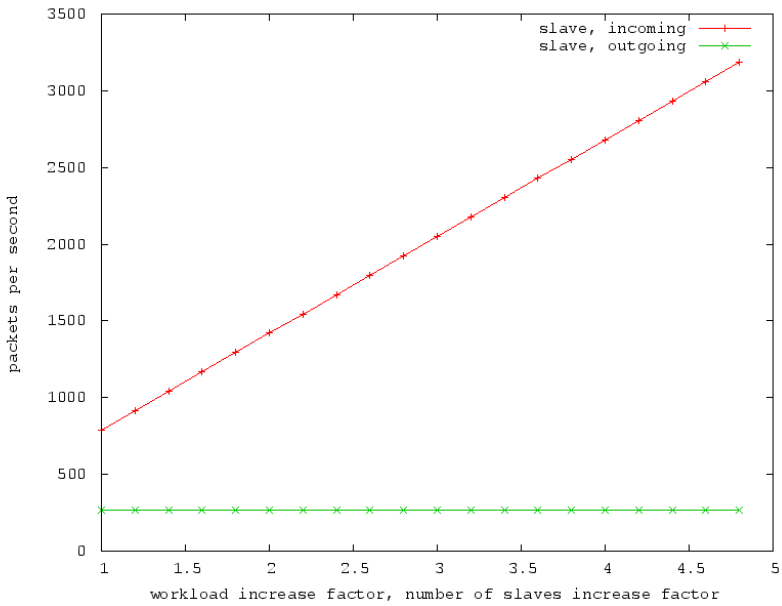
Fig. 7. Throughput of first master and slave databases

Figure 8(a) shows the effect of the same experiment on the incoming and outgoing network traffic of the first master node. Here, the y-axis contains the number of packets going in and out of the master node. The curve of the outgoing link shows the quadratic behaviour of this traffic, as number of outgoing packets is influenced by the increasing number of slaves and the increasing workload. The incoming traffic related to only the latter of those factors. Figure 8(b) shows the results of this experiment on the network link of a slave node. The x-axis and y-axis have the same properties as previously. The linear increase in incoming traffic is related to the incoming replication updates. The outgoing traffic remains constant, as this is the traffic to the clients. As more slaves are added proportionally to the increased workload, each slave handles the same number of requests from clients throughout the experiment. The analysis shows that all resources related to the replication of presence updates are contributing to the non-scalability of this architecture. The network links and the databases of both the database master node and the slave nodes overload when the workload is increased. Adding slaves only aggravates the problems at the master node. To see whether the new architectures have solved these scalability issues, similar experiments are performed on their models. We start by analysing the first, partition based architecture.

Figure 9(a) shows the throughput at the incoming and outgoing links of a slave node. The x-axis displays the number of slave nodes. We have chosen to increase the number of slave nodes per partition by ten for every increase of the workload factor. Because the number of partitions is kept constant at ten, the total number of slaves varies from one hundred to one thousand. The y-axis shows the number of



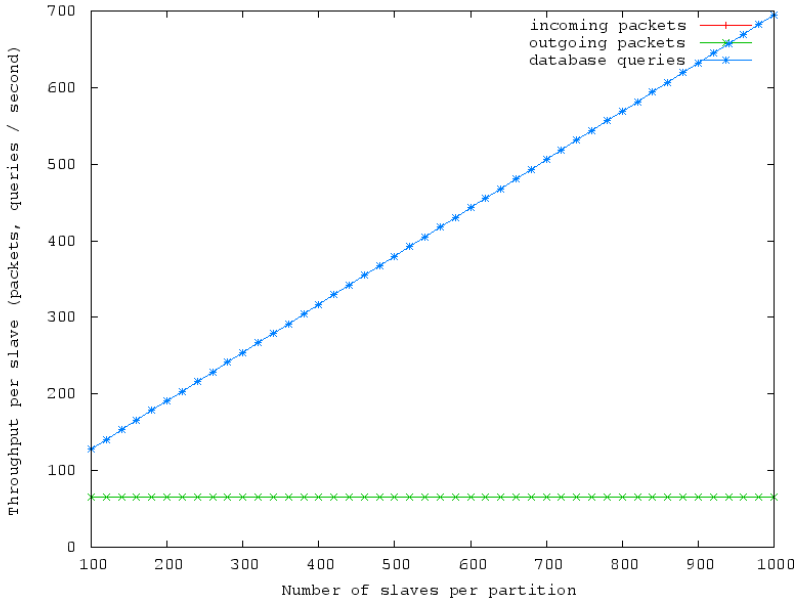
(a) Throughput of master network link



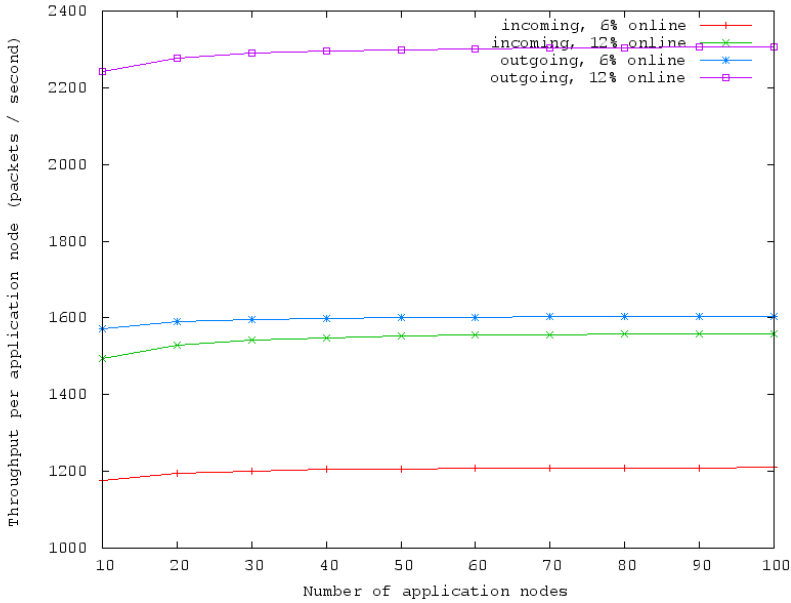
(b) Throughput of slave network link

Fig. 8. Scalability analysis of network traffic in the current architecture

network packets and database queries per second. The number of incoming packets is equal to the number of queries, which increases linearly with the workload. This shows that by itself, the increasing the number of slaves per partition does not solve scalability problems. The number of partitions has to be increased as well to limit the load per slave node. We also change other parameters than just the workload.



(a) Throughput of slave nodes

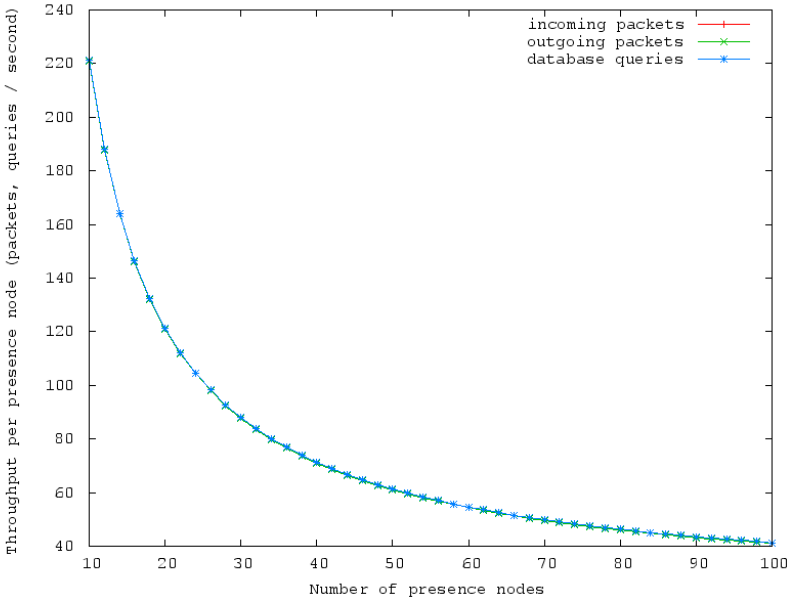


(b) Throughput of application nodes for different parameter settings

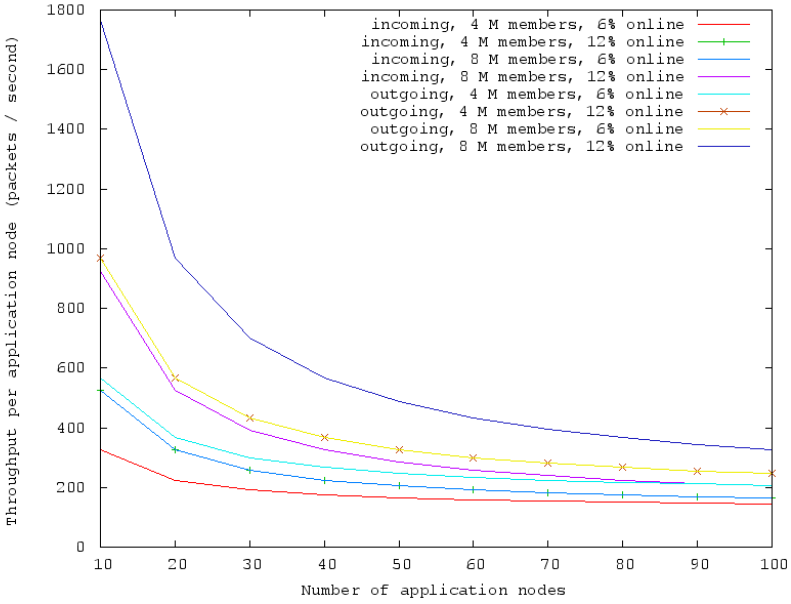
Fig. 9. Proportionally increasing loads in Architecture 1

Figure 9(b) gives the results for the application nodes for two different percentages of online users. The original peak percentage of online users is 6%. Doubling this value causes a slight increase in the load on the network links of the application nodes. For other node types, changing this parameter had no influence. We can conclude that the application and database master nodes of this architecture are

linearly scalable. However, scaling the database slave nodes requires adding both slaves to partitions and new partitions. This results in a quadratic increase in number of machines. Hence, this architecture does not scale linearly. We analyse



(a) Throughput of presence nodes



(b) Throughput of application nodes for different parameter settings

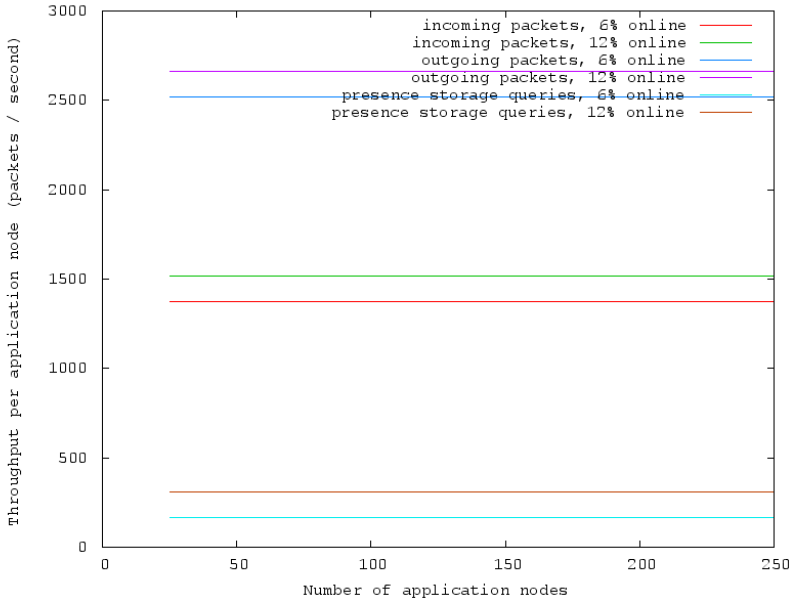
Fig. 10. Proportionally increasing loads in Architecture 2

the scalability of the second, aggregates presence architecture in a similar way as

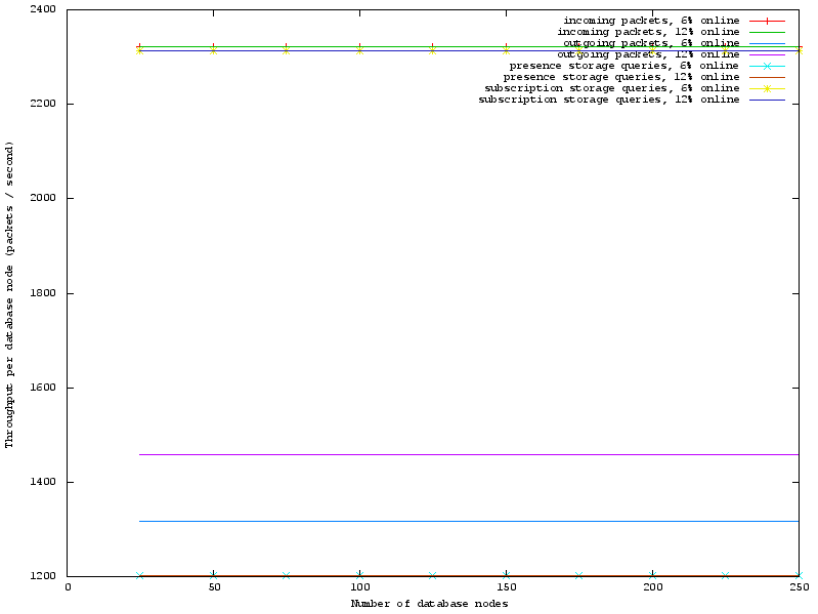
the previous architectures. Again, the workload and one architecture parameter are varied proportionally while all other architecture parameters are kept constant.

Figure 10(a) contains the throughput of each resource in a presence node. The x-axis shows the number of presence nodes, while y-axis shows the number of incoming and outgoing packets and the number of database queries. The workload increases from the current peak rates to ten times the current peak rates, proportionally to the increase in the number of presence nodes. The number of queries and the number of incoming packets are equal, as each incoming packet leads to a query to the database. The number of outgoing packets is slightly less as the model does not generate a response packet for incoming presence information. Again, these curves indicate that the load on this architecture is not dependant on the rates of the actions. While the workloads increase, the throughput of each presence node decreases. The experiment was repeated for doubled values of the total number of members and the percentage of online members. Figure 10(b) shows the results for the application node. It shows that doubling the number of members or doubling the online percentage has the same result for both incoming and outgoing traffic. Doubling both nearly quadruples the traffic. The graph shows that there is almost a direct relation between the two parameters and the throughput in the application node. Experiments on the presence nodes showed a similar result. On the other hand, the channel nodes were not influenced at all by changes in these parameters. The scalability properties of this architecture depend on more factors than the workload parameters, such as the number of online users. When these factors remain constant, as assumed, this architecture scales sublinearly. This means that this architecture is able to handle additional load without needing a linearly proportional number of extra machines. Similar to the other two proposals, we use our HIT model to analyse the scalability properties of the third, subscription-based. Again, we increase the workload linearly while we increase the number of nodes of one type proportionally.

Figure 11(a) gives the results for the application nodes. Starting with 25 application nodes and one time the current workload, we add 25 application nodes every time the multiple of the workload increases by one. The number of presence nodes is kept constant at 250. These numbers differ from the numbers used when analysing the other proposals, as this model contains more storages, requiring more nodes of both types for the same workload. The throughputs for the incoming and outgoing links are given in the y-axis. The curves of the links and storages are all constant. This indicates that this part of the architecture scales for increasing workloads. The traffic and storage queries in the application node increase by only a small amount when the number of online users in increased. Figure 11(b) shows the results of the corresponding experiment for the database nodes. Here, the number of application nodes is kept constant at 250 while the number of database nodes increases by 25. Each database node contains a separate storage for the presence information and the subscriptions. The graph shows that the subscription storage is accessed about two times as frequently as the presence storage. Each curve is constant, indicating good scalability of the database nodes. Only the throughput of the outgoing link



(a) Throughput of application nodes for different parameter settings



(b) Throughput of database nodes for different parameter settings

Fig. 11. Proportionally increasing loads in Architecture 3

increases slightly when twice as many users are online at the same time. The results of these experiments show that the components of this architecture scale linearly.

The analysis results have all been based on throughput and utilisation. We now analyse the architectures based on response time. The results of this analysis do not relate immediately to latency experienced by users, as the models do not contain all

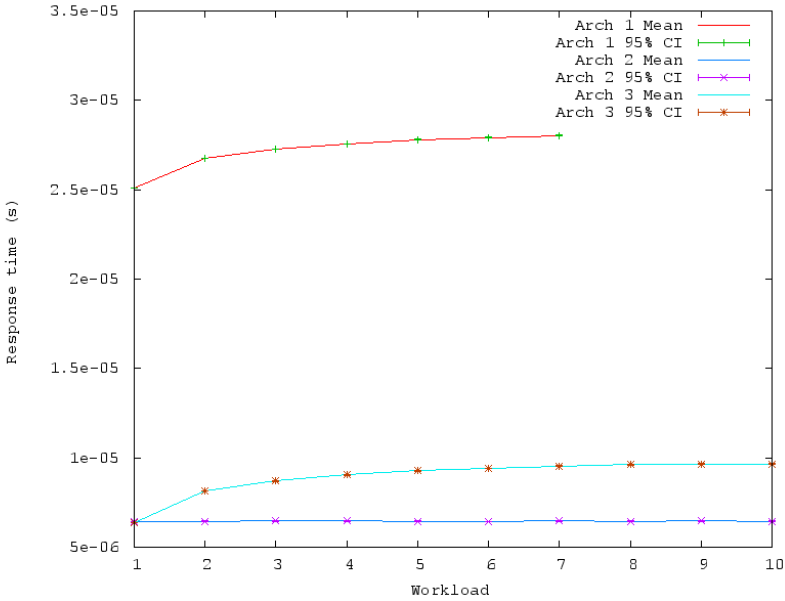


Fig. 12. Latency in sending instant messages for different workloads

aspects of real systems that determine latency. The analysis could however reveal fundamental problems of an architecture. For example, a constantly increasing response time indicates that the architecture will in practice exhibit performance problems. Architecture 2 delays presence updates by default, making it difficult to compare architectures on the response time of presence updates. Therefore, we only compare architectures on the response time of propagating an instant message.

Figure 12 shows the latency of sending an instant message in all architectures. The number of nodes is increased while more workload is applied. such that the load on each node remains constant. In case of Architecture 1, this means that a super-linear amount of nodes is added. Because of this, the analysis of more than seven times the current workload is impossible because of this increase. For both Architectures 1 and 3, the response time increases initially and converges to a fixed value. This relates to the fact that for an increasing number of application nodes, the probability that the message has to be transmitted from one application node to another converges to one. In other words, the probability that the sender and receiver of an instant message are connected to the same application node and no additional network traffic is needed decreases to zero as the number of application nodes increases. The difference in response time between the first and third architecture is caused by the fact that the first architecture needs a database lookup for each instant message, while the third does not. In the second architecture, an instant message always takes the same path and the response time is constant during the entire experiment. The graphs show that using each archicture it is possible to deliver an instant messaging service that has a predictable response time. No evidence of response time problems is found at this level of modelling detail.

workload	numap	numspp	numpar	total
1	7	4	6	37
2	23	7	21	191
3	51	9	41	461
4	102	12	72	1038
5	179	14	110	1829
6	286	17	153	3040
7	460	19	219	4840
8	625	22	264	6697
9	978	24	375	10353
10	1169	27	405	12509

Table 3
Number of machines needed in Architecture 1

workload	numap	numch	numpr	total
1	7	5	25	37
2	12	10	27	49
3	17	15	28	60
4	22	20	29	71
5	27	25	30	82
6	32	30	31	93
7	37	35	33	105
8	42	40	34	116
9	47	45	35	127
10	51	50	36	137

Table 4
Number of machines needed in Architecture 2

Finally, we compare architectures on the number of machines. The results of this experiment are given in Table 3, 4 and 5. The total number of machines is plotted in Figure 13. The tables show for each architecture the number of machines necessary for each separate type of node. The column names refer to the input parameters of the respective models. For Architecture 1, the relation between the number of

workload	numap	numdb	total
1	8	29	37
2	17	58	75
3	25	87	112
4	33	116	149
5	41	145	186
6	49	174	223
7	57	203	260
8	65	232	297
9	73	261	334
10	81	290	371

Table 5
Number of machines needed in Architecture 3

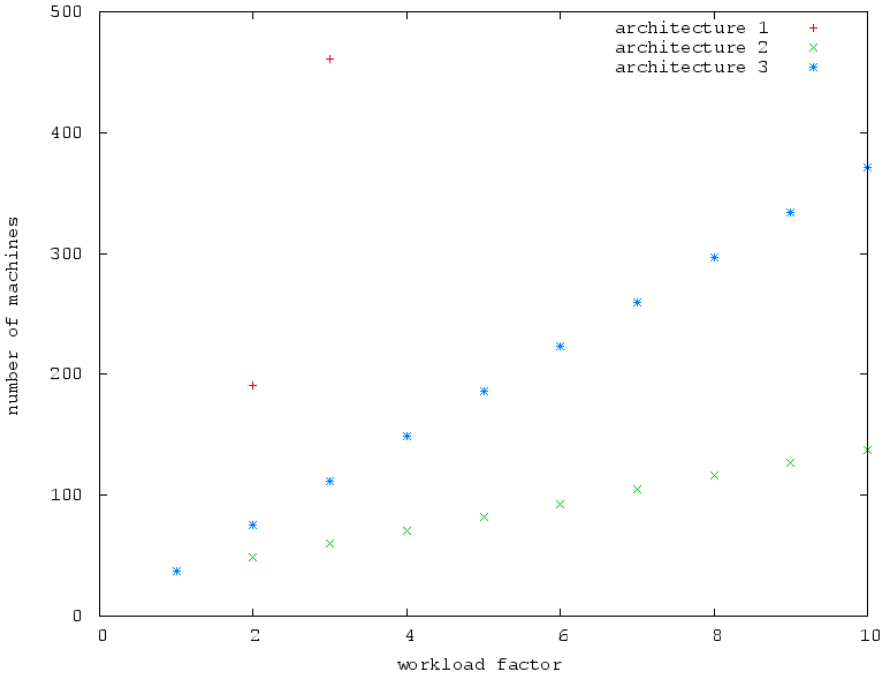


Fig. 13. Number of machines needed in the three different architectures

nodes is complex. For instance, when adding more slaves to cope with additional read load, the additional replication traffic increases the load on the master. To decrease this to the required level, partitions have to be added, increasing the load

on the application nodes. This complexity shows in the analysis results: there is clearly a superlinear increase in number of machines, but a relation between workload and machine count is hard to deduce.

The analysis shows that the second and the third architecture scale linearly. However, the third architecture the required number of machines is a constant multiple of the workload factor, while for the second architecture there is a base number of machines required, independent of the workload. This is deduced from number of presence servers (*numpr*), which remains fairly constant. This is a direct result from the aggregation of presence updates in this architecture. Recall that the load generated by the distribution of presence updates is unrelated to the number of presence updates per second. This property is unique for this architecture. From this we conclude that the first architecture is not scalable. The second architecture is scalable, with the reservation that the number of machines is only partially determined by the workload. The third architecture is scalable in the most strict sense of the word: the number of machines is related linearly to the workload.

6 Conclusions and future work

We introduce a suitable modelling and analysis approach for scalability analysis of IM&P architectures. It abstracts from the unnecessary performance aspects of the architecture and focuses solely on the relation between workload and the use of databases and network links. Analysis shows that the current architecture does not scale in almost all of its parts: the database master nodes do not scale as there is no possibility of adding machines to share the load. We found that the database component will be the first to reach its limit for increasing loads.

The third subscription-based alternative has a strictly linear relation between workload and utilisation. This relation is also linear for the second, aggregated presence architecture, but here the utilisation is only partially determined by the the workload. The first, partitioned alternative has the worst scalability in this comparison, its machine to workload ratio increases dramatically for larger workloads. The second architecture introduces delays in presence propagation and a limit on the amount of presence information that can be stored per user. These findings lead us to recommend the subscription-based architecture as the best architecture for Instant Messaging & Presence services. The evolutionary improved architecture might be used as a short term solution if scalability problems arise, but the model shows that it does not scale in the long run. The aggregated presence updates architecture is also scalable, but is less favourable, given the degradation in quality of service for presence propagation.

Future work on this topic can be done in a multitude of directions. On the practical aspect, comparing the performance of the architectures by simulation or prototyping can provide additional insight in the benefits of each architecture. Also, the precise resource requirements of the single architecture should be investigates for resource planning. On the theoretical side, better tool support for the analysis of computer architectures needs to be developed. HIT and HITGRAPHIC were

sufficient, but specific enhancements for models of scalable architectures would decrease the analysis effort. Specifically, modelling multiple identical components is only possible using simulation, while it could probably be analysed analytically if HIT would support this. The HITGRAPHIC representation of a HIT model makes modelling easier, but this can be further enhanced by focusing more on relations between components. Cycles in the component hierarchy are not allowed, which prevented a straight-forward implementation of the current architecture.

Acknowledgement

We would like to thank the group *Quantitative Techniques in Computer Science* at the University of Dortmund for providing access to and support for the Hierarchical Evalution Tool.

References

[1] L. A. Barroso, J. Dean, and U. Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(02):22–28, March 2003.

[2] H. Beilner, J. Mäter, and N. Weißenberg. Towards a Performance Modelling Environment: News on HIT. In *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 57–75. Plenum, 1989.

[3] H. Beilner, J. Mäter, and C. Wysocki. The Hierarchical Evaluation Tool HIT. In D. Potier and R. Puigjaner, editors, *Short Papers and Tool Descript. of 7th Int. Conf. on Modelling Techniques and Tools for Computer Perf. Evaluation*, pages 6–9, 1994.

[4] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proc. of the 2nd International Workshop on Software and Performance*, pages 195–203. ACM Press, 2000.

[5] M. D. Hill. What is scalability? *SIGARCH Computer Architecture News*, 18(4):18–21, 1990.

[6] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663. ACM Press, 1997.

[7] E. Letucky. Facebook Chat, May 2008.

[8] Y. Liu and I. Gorton. Performance Prediction of J2EE Applications Using Messaging Protocols. In *Component-Based Software Engineering*, volume 3489/2005, pages 1–16. Springer, 2005.

[9] M. Nicola and M. Jarke. Performance modeling of distributed and replicated databases. *Knowledge and Data Engineering, IEEE Transactions on*, 12(4):645–672, 2000.

[10] C. Piro. Chat Stability and Scalability, Feb 2009.

[11] D. Reisinger. How Twitter could be worth nothing in a year, Jul 2008.

[12] J. Schippers. A Massively Scalable Architecture For Instant Messaging & Presence. Master's thesis, University of Twente, July 2009.

[13] L. P. Slothouber. A Model of Web Server Performance. In *5th Int. World Wide Web Conference*, 1995.

[14] C. Torre. Windows Live Messenger - What. How. Why., Jul 2006.