A Massively Scalable Architecture For Instant Messaging & Presence

Jorrit Schippers\textsuperscript{a}, Anne Remke\textsuperscript{a}, Henk Punt\textsuperscript{b}, Maarten Wegdam\textsuperscript{c}, Boudewijn Haverkort\textsuperscript{a}

\textsuperscript{a}Design and Analysis of Communication Systems, University of Twente
\textsuperscript{b}Hyves
\textsuperscript{c}Novay

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 of alternative architectures. 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.

Key words: Instant Messaging and Presence Architecture, Queueing Models, Scalability

1. Introduction

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 instantaneous 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 aim of this paper is to determine a scalable architecture for an IM&P service, with the Hyves architecture as use-case. We take a queueing network-based modelling and
analysis approach to find the bottlenecks of architectures. Furthermore, we present architectures that provide better scalability.

**Related work.** Although several high level descriptions of IM&P architectures are available on the internet, there is no previous work on analysis of these architectures. However, a number of studies of architectures with other purposes have been published. The goal of most of them was to develop an analytically solvable model for an existing computer or software architecture. In [1] a simple webserver architecture is modelled and analysed using queueing networks. Using this model the benefits of several changes to the architecture, such as faster links or faster servers, were analysed.

A general approach to analyse distributed architectures using Markov chains and queueing networks is given in [2]. It involves modelling the system using a Discrete Time Markov Chain (DTMC). The resulting total service requirements are used to create a closed Product Form Queueing Network (PFQN). The PFQN is processed by the SHARPE software package, to find bottlenecks and estimate the merits of architectural modifications.

A comparison between two Enterprise Java Bean architectures using queueing networks is performed in [3]. A client - server architecture is studied, where the server is composed of a front-end request handler and a back-end message based request processor. The input variables for the queueing stations are determined using benchmarking on a physical implementation. The error margin between measured and predicted performance measures proved to be within 15%.

A similar problem is discussed in [4], which investigates the concurrency limits of an Enterprise Java Bean container. The components of the architecture are modelled separately using Markov Chains. The models are aggregated in a Flow Equivalent Server (FES) model. The FES model is compared to a simulation model that was created with the tool QNAP2. The model very well approximates both configurations.

The SIP Instant Messaging and Presence Leveraging Extensions working group of the Internet Engineering Task Force develops an IM&P extension to the Session Initiation Protocol (SIP). The working group has published an analysis of the message load caused by increasing workloads in various inter-domain scenarios [5]. Inter-domain presence exchange uses a publish/subscription mechanism similar to the third alternative architecture discussed in this paper. Although the paper does not discuss architectural properties, the presented numbers indicate that the presence extensions for SIP will cause a high load in terms of number of messages, network bandwidth, state management and CPU. The authors conclude by recommending research on a optimised server-to-server protocol for inter-domain scalability.
Structure of the paper. This paper is structured as follows: we will detail our modelling and analysis approach in Chapter 2. In Chapter 3 we describe the current IM&P architecture and its bottlenecks. Chapter 4 contains detailed descriptions of three alternative architectures. In Chapter 5 we describe the analysis results for the current and alternative architectures. Chapter 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 HITGRAPHIC [6, 7]. 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 propagates presence with a delay of up to five minutes [8, 9], which 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. All configurable parameters of the model are increased proportionally to the workload, as the response time is influenced by all parts on an architecture. HIT features necessary to analyse the response time of the architectures are only available when the model is solved by simulation. Without modifications, simulating large numbers of nodes was impossible. As a remedy, we have increased the service rate of each database component to be able to simulate the models using lower amounts of nodes.

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 system. They handle connections from clients, interpret requests, call external services or other nodes to fulfill 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 [10]. This means that the current architecture is not scalable: the architecture is not able to increase its capacity linearly by adding more hardware.

4. Architecture proposals

4.1. Architecture 1: Evolutionary partitioning

The first architecture proposal was developed by applying partitioning to the current architecture. Figure 2 shows the new composition of the components in this architecture. The architecture contains application nodes that have the same role as in the current 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 to a partition by a partitioning algorithm such as Consistent Hashing [11]. 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 partitions 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 second proposal is inspired by the Facebook chat architecture [8, 9], which serves over 200 million Facebook members. Figure 3 gives an overview of an 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.

![Figure 3: Overview of the aggregated and batched presences approach](image)

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 statuses 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

The third architecture proposal is inspired by the architecture of Windows Live Messenger, which is a popular IM&P service with over 450 million members [12]. Figure 4 gives an overview of 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.

![Figure 4: Overview of subscription-based architecture](image)

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 5 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.

Figure 6(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 6: Scalability analysis of network traffic in the current architecture
Figure 6(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 7(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 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.

In addition to the workload, also other parameters are changed. Figure 7(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 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 7: Analysis of proportionally increasing loads in architecture proposal 1
Figure 8: Analysis of proportionally increasing loads in architecture proposal 2
Figure 8(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 8(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 appear to 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 9(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 9(b) shows the results of the corresponding experiment for the
Figure 9: Analysis of proportionally increasing loads in architecture proposal 3
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 increases slightly when twice as many users are online at the same time.

The results of these experiments prove that the components of this architecture scale linearly.

Figure 10: Analysis of message sending latency

Figure 10 shows the latency of sending an instant message in all three architecture proposals. The x-axis shows the increasing workload factor, which is the same as each architecture parameter. The y-axis shows the latency in seconds, but this is just the abstract latency caused by the models of the network links and databases. The graph shows that for the first and third proposal the latency increases, but converges to a constant value after a while. This is related 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. Architecture 3 is faster, as this architecture does not need a network look up for each instant message. The
graph shows a constant value for the message sending latency in the second architecture proposal. This can be explained by the fact that the process of delivering an instant message does not depend on the number of nodes: a message will always be delivered at the channel node of the recipient. As the number of these nodes increase linearly with the load, the latency does not change.

These results indicate that each architecture is able to deliver messages with a constant delay. Only the first architecture needs a super-linear amount of nodes, as both the number of partitions and slaves per partition are increased. This also makes it impossible to simulate workloads over 7, as the total number of nodes becomes too large for the HIT simulator.

6. Conclusions and future work

We introduced a suitable modelling and analysis approach for scalability analysis. 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. Using this approach, we found 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.

Comparative analysis shows that 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 has the downsides of delays in presence propagation and a limit on the amount of presence information that can be stored per user. Latency of message propagation is acceptable in all architectures, as adding machines will not cause the latency to increase infinitely.

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 is possible in a multitude of directions. On the practical aspect of this paper, comparing the performance of the architectures
by simulation or prototyping can provide additional insight in the benefits of each architecture. Also, research can be aimed at just one architecture and discover the precise resource requirements of this architecture to aid 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 could 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.

7. Acknowledgements

We thank the University of Dortmund and especially Jürgen Mäter for providing access and technical support for the Hierarchical Evaluation Tool.

References

[1] L. P. Slothouber, A Model of Web Server Performance, in: 5th Int. World Wide Web Conference, 1995.

[2] V. S. Sharma, P. Jalote, K. S. Trivedi, Evaluating Performance Attributes of Layered Software Architecture, in: Component-Based Software Engineering, Vol. 3489, Springer, 2005, pp. 66–81.

[3] Y. Liu, I. Gorton, Performance Prediction of J2EE Applications Using Messaging Protocols, in: Component-Based Software Engineering, Vol. 3489/2005, Springer, 2005, pp. 1–16.

[4] C. M. Llado, P. G. Harrison, Performance evaluation of an enterprise JavaBean server implementation, in: Proc. of the 2nd Int. Workshop on Software and Performance, ACM, 2000, pp. 180–188.

[5] A. Houri, E. Aoki, S. Parameswar, T. Rang, V. Singh, H. Schulzrinne, Presence Interdomain Scaling Analysis for SIP/SIMPLE, Internet Draft (October 2008).
URL http://tools.ietf.org/html/draft-ietf-simple-interdomain-scaling-analysis-05

[6] H. Beilner, J. Mäter, N. Weißenberg, Towards a Performance Modelling Environment: News on HIT, in: Modelling Techniques and Tools for Computer Performance Evaluation, Plenum, 1989, pp. 57–75.
[7] H. Beilner, J. Mäter, C. Wysocki, The Hierarchical Evaluation Tool HIT, in: D. Potier, R. Puigjaner (Eds.), Short Papers and Tool Descript. of 7th Int. Conf. on Modelling Techniques and Tools for Computer Perf. Evaluation, 1994, pp. 6–9.

[8] E. Letucky, Facebook chat (May 2008).
URL http://www.facebook.com/note.php?note_id=14218138919

[9] C. Piro, Chat stability and scalability (Feb 2009).
URL http://www.facebook.com/note.php?note_id=51412338919

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

[11] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, 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, ACM Press, 1997, pp. 654–663.

[12] C. Torre, Windows live messenger - what. how. why. (Jul 2006).
URL http://channel9.msdn.com/posts/Charles/Windows-Live-Messenger-What-How-Why/