# Performance evaluation on GlobeCon - a scalable context management framework

Kaiyuan Lu, Doron Nussbaum and Jörg–Rüdiger Sack

School of Computer Science, Carleton University
{klu2,nussbaum,sack}@scs.carleton.ca

**Abstract.** Designing a large scale context aggregation and provision system that integrates distributed, heterogeneous and disparate context providers and consumers is a challenging problem. As a system grows numerically or expands geographically, functional building blocks can quickly become bottlenecks in terms of reliability as well as performance. In this paper, we present performance evaluation results of our context management system, GlobeCon, which we have designed recently[19]. GlobeCon is a scalable framework that supports distributed context collection, aggregation, processing, provision and usage in large scale ubiquitous computing environments. We present a set of evaluation criteria and matrices: *i.* overhead; *ii.* effectiveness; *iii.* adaptability; and *iv.* scalability/throughput, which are used for measuring the overall performance of GlobeCon. We achieved very good results in performance as well as resource utilization. Experimental results show that GlobeCon hierarchical design achieved 100% throughput under the load of 15,000 msg/sec with 1500 concurrent sensors, and 1000 concurrent consumers. Load tests on GlobeCon, which tested the limits of some of the components, show that a LoCoM can achieve 90% throughput under the load of 300,000 msg/sec. Moreover the message delay under extreme load is only tens of milliseconds and remains constant throughout the tests.

## 1 Introduction

Designing of context management system/framework, which supports effective context collection and dissemination, is one of the essential tasks to realize context awareness computing. A context management system can be designed and implemented in a variety of ways depending on different requirements and deployment environments. In this paper, we are interested in providing context management within large scale environments that integrate participants and resources of various kinds spanning outside physical areas. A system like this requires additional design considerations besides fundamental function modules (i.e., context acquisition, representation, processing and accessing).

The additional design considerations, as summarized in [19], encompass the following four aspects. *Scalability* exists in three dimensions: numerical (i.e., the increase of workload of a single resource), geographical (i.e., the expansion of a system from concentration in a local area to a more distributed geographic pattern) and administrative (i.e., the coexistence of a variety of software/hardware

platforms in a system). It refers to the ability to handle the growing demand in any of the three dimensions in a graceful manner without degrading performances. *Distributed nature* is introduced by the large number of raw context sources and context processing resources that are spread over large geographic areas. *Dynamic nature* is inherent in context sources and users which are mobile and changing their states, e.g., a sensor moves from place to place and switches its status between active and inactive, and in the physical environment of pervasive computing, e.g., the traffic condition of a road network. *Heterogeneity* is arisen due to the need in dealing with multiple data sources (i.e., context sources with heterogeneous hardware/software interfaces and raw context data in different formats). In our previous work, we proposed the design of GlobeCon, a hierarchical context management framework that aims at supporting distributed context aggregation, processing, provision and usage in large scale ubiquitous computing environments. GlobeCon addresses the above mentioned design considerations with regards to large-scale context management systems.

In this paper, we focus on evaluating the real applicability of GlobeCon from experimental point of view. We believe measurement studies of this kind are of essential practical importance, since they provide guidance for the design of real-life systems, help to pinpoint potential bottlenecks and give insight on approaches for improvements. We design a set of evaluation metrics in terms of system maintenance overhead, effectiveness, adaptability and scalability/throughput, which are applied to GlobeCon. Our experimental results demonstrate that GlobeCon scales well in a number of scenarios, and it is able to efficiently manage context producers and consumers in highly dynamic environments with little overhead.

The rest of this paper is organized as follows. Section 2 reviews related work in literature. Section 3 gives an overview of GlobeCon architectural design and its implementation details. Section 4 illustrates the set of performance metrics we used, and elaborates on our experiments and analysis on the results. Section 6 concludes the paper.

## 2 Related Work

Growing rapidly since the Active Badge project [26], research on context aware computing has attracted a lot of attention as witnessed by a large body of literature. In this paper, we concentrate our work and literature review on context frameworks or middleware. For a more comprehensive discussion we refer the interested reader to recent survey articles [6] [20][24].

Context Toolkit [9] was one of first and influential projects that emphasized the isolation of application logic from the details of context sensing and processing. A conceptual context handling framework, based on a centralized component (i.e., the discoverer), was proposed. This framework provides context gathering, interpretation, and aggregation support for applications. CASS [10] uses a middleware approach to support context awareness for hand-held computers. The middleware resides on a centralized resource-rich server. CASS supports context acquisition, interpretation, and reasoning. CMF [18] is a stand-alone software framework designed for mobile terminals. It supports acquisition and processing of contexts in a user's surroundings, and gives the contexts to applications

residing on the user's mobile device. CoBrA [4] is a broker centric agent-based infrastructure supporting context aware computing in dynamic smart spaces such as an intelligent meeting room. It has a central context broker which provides four functional modules: context acquisition, context knowledge base, context reasoning engine and context privacy control.

The above mentioned frameworks have in common a centralized component in their design, which may become a bottleneck when the number of context sources (i.e., sensors) and context consumers increases dramatically. Moreover, they do not scale well in the geographical dimension. SOCAM [14] is service-oriented context aware middleware architecture, which addresses the scalability problem with a global-wide Service Locating Service (SLS) [15]. SOCAM supports context acquisition, discovery, interpretation and context access through a set of independent services, which advertise themselves with the SLS. SCI [13], a generalized context framework, tries to address the scalability problem with a two-layer infrastructure, where the upper layer is a network overlay of partially connected nodes (named Ranges), and the lower layer concerns the contents of each Range in the overlay network. A Range provides context handling services within an area. However, the authors did not mention how to create an overlay network to support the interactions among Ranges, which in our opinion is one of the key challenges in solving the geographical scalability. Similarly, Solar [5] addresses the scalability issue using overlay networks, but does not provide a way to construct such an overlay network. In CoCo [3], the authors argued to utilize the existing infrastructures (i.e., grids, peer-to-peer networks, and Content Delivery Networks) to support scalability in a context management framework. Based on this assumption, CoCo provides an integration layer, which interfaces with heterogeneous context sources, maps the retrieved context to a standard information model, and provides it to context consumers in a unified way.

Our design of GlobeCon achieves scalability by introducing a hierarchical management structure, a dual resource discovery and various optimization for load balancing. Most importantly, our work appears as the first work in literature that provides an extensive set of simulation results in terms of system maintenance overhead, throughput, adaptability and scalability. We believe our measurement study can provide guidance for the design of real-life systems.

## 3 GlobeCon

### 3.1 Architecture Design

GlobeCon is designed as a scalable hierarchical context management system. It is aimed at supporting distributed context aggregation, processing, provision and usage in large scale pervasive computing environments. Fig. 1 depicts the architectural design of GlobeCon and its main building blocks. Three key design features enable scalability in the numerical, geographical and administrative dimensions (refer Section 1.): (i) organizing context aggregation in a hierarchical manner (i.e., Sensors, Sensor Managers, Locol Context Managers and Universal Discovery Service), (ii) effectively distributing and balancing the load of context

collection and processing (i.e., interpreting and reasoning) among Local Context Managers (LoCoM), and (iii) dual resource discovery mechanisms: proximity-based discovery for raw context sources and Universal Discovery Service (UDS) for discovering the largely distributed LoCoMs.

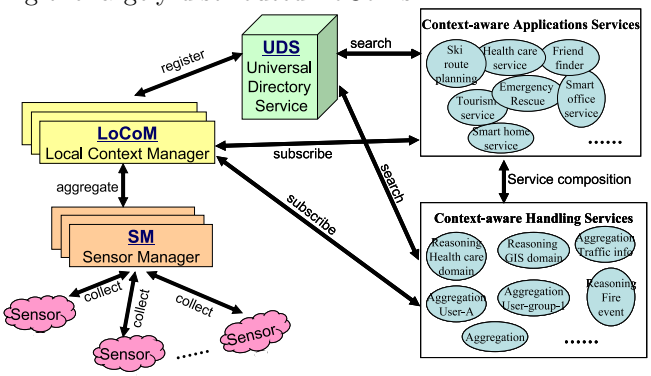



**Fig. 1.** An overview of GlobeCon architecture and its main building blocks

**Sensors** The sensor level comprises the ground level context sources, which can be in the form of hardware or software. Sensors have three dominating characteristics: highly constrained resources (e.g., CPU, memory), high mobility, and heterogeneous interfaces. Due to the limited resources, sensors are designed to perform very simple tasks: capturing raw context from the environment and transmitting the data to the Sensor Managers (SeMs).

**Sensor Managers** A SeM acts as a gateway between sensors and LoCoMs. It gathers raw context from registered sensors, and manages the heterogeneities of different sensors. Once the data is recieved, the SeM relays the data (possibly after some manipulation and aggregation) to LoCoMs that it is registered with. A SeM interacts with context consumers as well to reply their context queries.

**Local Context Managers** A LoCoM is designed to perform computation and communication intensive functions within the context processing flow (e.g., detecting/resolving context correlation and inconsistency, and context reasoning). It interacts with SeMs to aggregate context, interacts with UDS to register itself and update the UDS of any changes on the context types that it can currently offer, and interacts with context consumers to listen and reply context queries. Due to its functional design, we envision that it should be hosted by high performance computing devices with high bandwidth network connections.

**Universal Directory Service** UDS plays two main roles in the GlobeCon framework: first, it is a global registry, storing all meta data associated with LoCoMs, including their context types, how to reach and make use of them; second, it is a search tool, which locates a LoCoM that provides context information required by a context consumer. The UDS is composed of multiply distributed physical servers which can be viewed as one logical box providing the service.

**Service Groups** Two groups of services depicted in Fig. 1, are context consumers that utilize the context aggregated and provided by SeMs and LoCoMs. Context handling services are various high level, large scale context handling services (e.g., context reasoning, context integration), which are usually application

domain specific. Context aware applications/services are a variety of ubiquitous services that are context dependent and aimed for end-users.

### 3.2 Implementation details

As discussed in Section 1, GlobeCon, as a context management system for large-scale pervasive computing environments, is required to handle scalability, dynamicity, distribution and interoperability. Therefore, the GlobeCon's main building blocks, with the exception of the UDS (top of the hierarchy), are loose coupled from each other, and are distributed and pluggable. In this way, mobile components like context sensors, SeMs and context consumers can enter and leave the GlobeCon run time environment at will, based on their mobility and resource conditions. Furthermore, SeMs and LoCoMs have the flexibility to coordinate with each other to dynamically balance their loads (i.e., a LoCoM may delegate SeMs to neighboring LoCoMs; or it can adjust sensor's sampling rate on-the-fly), and thus avoid any one to become a bottleneck when the system is growing in size (e.g., when the number of context producers/consumers is increased). In this section, we provide some insight into the implementation of GlobeCon's main components and their interaction.

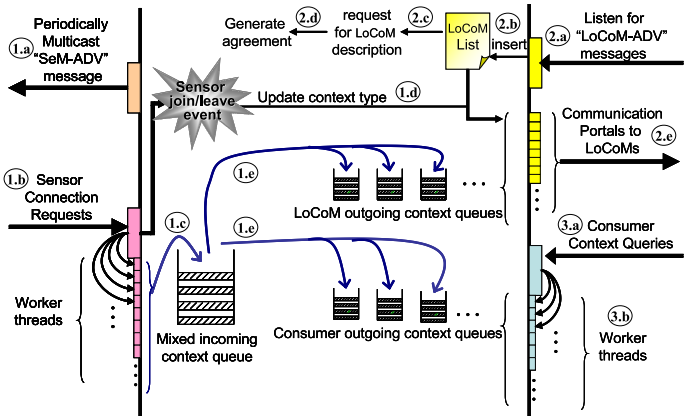



**Fig. 2.** The internal processing logics of a Sensor Manager

Fig. 2 depicts a simplified internal processing logic of a SeM and its interaction with sensors, LoCoMs and context consumers. For its interaction with sensors (i.e., context sources), (1.a), a SeM periodically broadcasts its availability over its wired/wireless LAN via "SeM-ADV" message, using a specified multicast group. Meanwhile, (1.b), it listens to connection requests from sensors and assigns a worker thread to handle each individual request according to the Sensor-SeM handshake protocol depicted in Fig. 4. The raw contexts received from all registered sensors (1.c) are then deposited at a local incoming context queue. For its interaction with LoCoMs, a SeM (2.a) listens "LoCoM-ADV" messages and (2.b) inserts each available LoCoMs into its local LoCoM list. It then (2.c) requests self-description from LoCoMs, based on which (2.d) an individual agreement is generated for each LoCoM. After that it allocates an outgoing context queue for each LoCoMs, and (2.e) sets up context data transfer session with the LoCoMs. A SeM also (3.a) has a server thread listening for context queries

from consumers, and (3.b) assigns a new thread to handle the interactions with each consumer concurrently. Raw context placed in the SeM's incoming context queue (1.e) will then be dispatched to corresponding LoCoM's/consumer's outgoing queues based on their respective agreements. In step (1.e), there are multiple dispatching threads serving the incoming context queue. Depending on current workload, the number of dispatching threads can be adjusted on-the-fly to dynamically balance workload and tune up SeM's performance. Upon the arrival of a sensor join/leave event, the SeM (1.d) will accordingly update LoCoMs and re-negotiate their agreements if necessary.

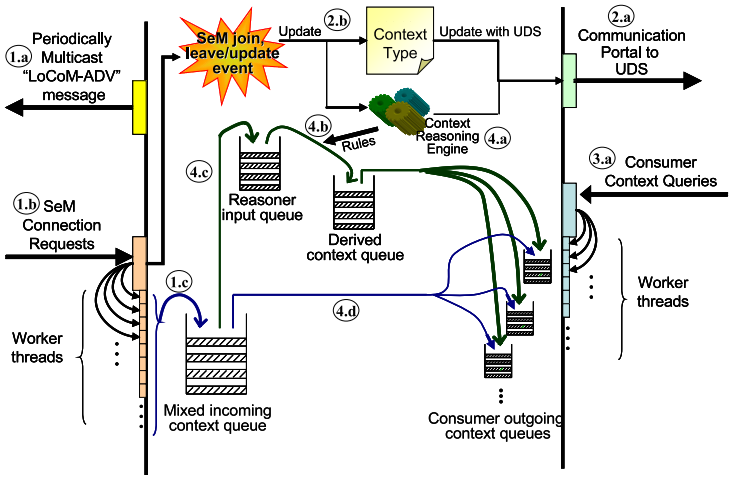



**Fig. 3.** The internal processing logics of a Local context manager

Fig. 3 illustrates the internal processing logics of a LoCoM and its interactions with SeMs, UDS and context consumers. Similar to SeM, (1.a) a LoCoM periodically broadcasts its availability to SeMs, and (1.b) listens to SeMs' connection requests, which are classified into two types: description request and data transfer session request. All contexts from SeMs (1.c) are temporarily stored at a mixed incoming context queue. Meanwhile, a LoCoM (2.a) registers itself with the UDS, which is a well know service within the GlobeCon run time environment. It also (3.a) has a portal for context consumers and allocates a queue for each consumer respectively. Inside of a LoCoM, there is (4.a) a rule-based context reasoning engine, which is able to (4.b) derive higher level context based on raw context. Context in the mix queue (4.d) will be dispatched to corresponding outgoing queues based on consumers' agreements; and some (4.c) goes to the reasoner input queue according to the knowledge base of the reasoning engine. Upon the arrival of a SeM join/leave/update event, the LoCoM (2.b) will update this information with the UDS and its local reasoning engine.

UDS exhibits great similarities with UDDI [8] and is actually a subset of UDDI in terms of functionality and infrastructure. There already exist several UDDI implementation and performance studies [2][23] (UDDI is a proven feasible and scalable discovery mechanism). Therefore, we decided to not focus our efforts

on the evaluation of UDS. For the GlobeCon prototype, we used a light weighted UDS implementation to make the whole system functioning.

The programming language, Java, was determined by these two considerations: (a) Java is portable across a variety of platforms; (b) one of our goals is to integrate the GlobeCon with our existing IMA system[1] to provide pluggable context management services, and IMA is implemented using Jade [16], which is a Java based agent development framework.

## 4 Experimental Studies

### 4.1 Evaluation Metrics

To study GlobeCon from an experimental perspective, we employ several metrics as stated below. We then evaluate and compare the performance and effectiveness of GlobeCon under different scenarios using these metrics.

**Overhead**: is defined as the resource consumption that GlobeCon components. We use CPU usage and memory usage as measurements. That information (i.e., native Java process size and a running process's CPU usage in our experiments) can be mined only from the operating system directly, which is beyond the scope of standard Java portability. Therefore, it is worth mentioning that we use native OS-dependent system calls written in C++ to get information and transfer data to Java through JNI (Java Native Interface).

**Effectiveness** (or context delivery delay): is defined as the time difference between the generation time of a piece of context at its source and its arrival time at SeMs, LoCoMs or context consumers. It measures one aspect of context quality from end user's point of view. The smaller the time difference, the more effective the system and the more accurate the context information is for end users. This time measurement includes communication delay and local context processing time at SeMs or/and LoCoMs, but not includes the time for system self-configuration (i.e., discovering context source in dynamic environments). Since this measurement involves timing references at different hosts, we need to take into consideration the asynchronous nature of different hosts in order to collect accurate statistics. A simple method employed here to offset this time drift, is to compute their local time difference of two hosts using a round trip communication.[2] The accuracy of this method is based on an assumption that the calculated time drift is accurate. We calculate this time under minimized network traffic conditions, therefore the assumption is justified.

**Adaptability** (or latency for dynamic system configuration): aims at measuring the time taken by SeM/LoCoM between the discovery of a sensor/SeM and its entering into a successful connection state with the sensor/SeM (i.e., ready for context data transfer). This measurement becomes increasingly relevant in a highly dynamic environment where context sensors, SeMs and context consumers enter and leave the GlobeCon run time environment frequently. The

[1] IMA [11] stands for Intelligent Map Agents. It aims at providing personalized and context-awared access to, and support for manipulating of, spatial information.

[2] Alternatively, we can synchronize the hosts with the NRC atomic clock [17].

shorter the measurement is, the better adaptability the system can achieve. This measurement is broken down into four evaluation metrics for sensor-SeM and SeM-LoCoM communication, respectively as shown in Figure 4.

**Throughput**: represents the ratio of the number of context items that are successfully handled and delivered to context consumers over the total number of context items that are received from context sources. Recall that a SeM/LoCoM temporarily stores received context at an incoming queue and dispatches them to consumers' outgoing queues waiting for network delivery. As workload increases, SeM/LoCoM eventually reaches a point where it can no longer handle all context in its queues. In such situations, to avoid overflow or stale context information[3], a SeM/LoCoM just removes some context items from the queues based on FIFO. *Throughput* is employed here to statistically measure this behavior.

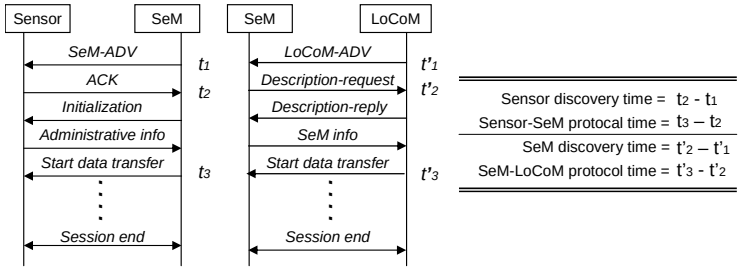



**Fig. 4.** Simplified Sensor-SeM-LoCoM protocol

### 4.2 Experimental Setup and Methodology

Experimental evaluation of a large scale distributed system like GlobeCon requires not only an implementation but also the examination of a large number of test scenarios. Our strategy is to divide the test scenarios into two groups. In other words, there is one group of experiments keeping track of the performance for SeM and LoCoM, respectively. Another consideration is to be able to evaluation SeM and LoCoM in terms of scalability. Therefore, we devised two sets of experiments for each group: (i) increasing the amount of workload imposed on a SeM and LoCoM (e.g., setting high sampling rate for context sources), (ii) increasing the network size of GlobeCon run time environment (i.e., increasing the number of context sensor, the number of SeMs and the number of context consumers). We keep track of the system maintenance overhead, effectiveness, adaptability and throughput under varied scenarios.

Another issue worthy mentioning is that there are a number of parameters (i.e., number of sensors/context consumers, sensor's sampling rate, SeMs' and LoCoMs' queue size) that have impacts on the overall system performance. Therefore, it becomes necessary to choose the proper parameters so that the experimental results will provide insights into the system behavior under various conditions. Our methodology, in wisely determining the selection of parameter as ranges, is a two step approach. (i) We first chose a large range for each parameter so that the critical points (i.e., points around which produce more fluctuant

[3] A better solution is to let SeM/LoCoM adjust the sources' sampling rate on-the-fly.

results) would be included within the ranges. (ii) Based on the analysis of the results of the first step, we then define refined parameter ranges to have a closer observation around the critical points. In subsequent sections, only the results of step-ii (using the fine-grained parameters) are presented.

We conduced our experiments using the HPCVL Beowulf cluster at Carleton University. The cluster consists of 64 nodes with 2x2.2 GHz Opteron Cores w/ 8 GB RAM per node. All nodes are interconnected via a Foundry SuperX switch using Gigabit ethernet. With the During the simulation, we confront the fact that at the beginning of each simulation run, the GlobeCon run time environment is not in a stable state because some of the system components (i.e., sensors, SeMs, LoCoMs and consumers) have engaged in their execution, while others are still inactive. Similar situation holds for the end of each simulation. To avoid distorting our measurements caused by this phenomenon, we only take into account the statistics during GlobeCon's stable state.

### 4.3 Experimental Results

**Performance of SeM** In this set of experiments, we studied the SeM's performance under increasing amount of work load ranging from normal to heavy. We fixed the number of SeMs to one and the number of consumers to 1000 so that we can analyze the impact of two parameters: the number of sensors and sensors' sampling rates. The number of context sensors ranges from 500 to 3000 and context sensor's sampling rate (SR) are 1Hz, 10Hz and 100Hz, respectively. Each simulation runs about 30 mintues to guarantee that our experimental readings are consistent. Within the SeM, each received context items is subscribed to by an average of 3 context consumers. In other words, the transmission load of a SeM's out-port is 3 times the load of its in-port under the condition that the SeM has zero drop rate. Three SeM's run-time parameters are set as follows: incoming queue size=5000, outgoing queue size=1000 and number of dispatching threads=400. In some cases, these SeM parameters are fine tuned so that we can better determine performance impacts of different combination of parameters.

Fig. 5 shows SeM's maintenance overhead in terms of CPU and memory consumption. As we can see, both CPU and memory consumption steadily increase as the workload grows in size from 500 concurrent sensors to 3000. Under heavy workload (i.e., 3000 concurrent sensors with SR=100Hz), the average CPU usage is 56.83%. Regarding memory usage, more sensors indicate more threads are created, which in turn means an increasing amount of thread stack memory allocation. The results of average memory usage give a trend as expected: 70.88MB for 500 concurrently connected sensors, 124.08MB for 1500 concurrently connected sensors and 206.23MB for 3000 concurrently connected sensors.

Fig. 6 shows the results of sensor discovery and SeM-Sensor protocol time. While measuring this metric, new sensors are gradually injected into the stable GlobeCon run-time environment, and statistics for the new sensors are collected. The discovery time actually closely depends on three factors (a) the interval $T_I$ between two consecutive *"SeM-Adv"* messages, (b) network latency and (c) SeM's current workload. $T_I$ is a constant set to 100 milliseconds throughout our

simulations. Fig. 6 (left) shows that the average sensor discovery time is slightly greater than 100/2 milliseconds, and is not sensitive to the growing of network size. Since the sensor-SeM protocol phase involves several handshake messages between sensor and SeM, the protocol time is proportional to the corresponding network latency as expected. Under heavy workload, SeM's responsiveness decreases, which in turn has great impact on SeM-Sensor protocol time as demonstrated in Fig 6 (right) (i.e., SR=100Hz and the number of sensors reaches 3000).

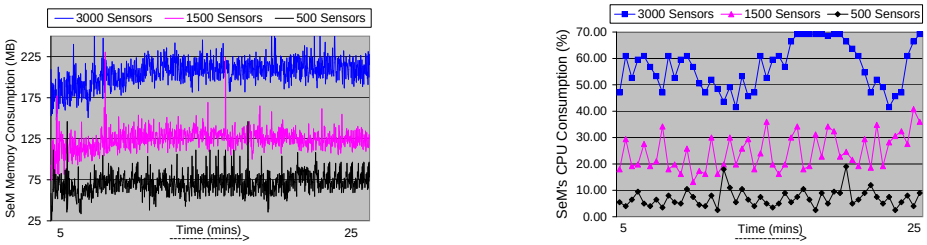



**Fig. 5.** SeM memory and CPU usage, SR=100Hz

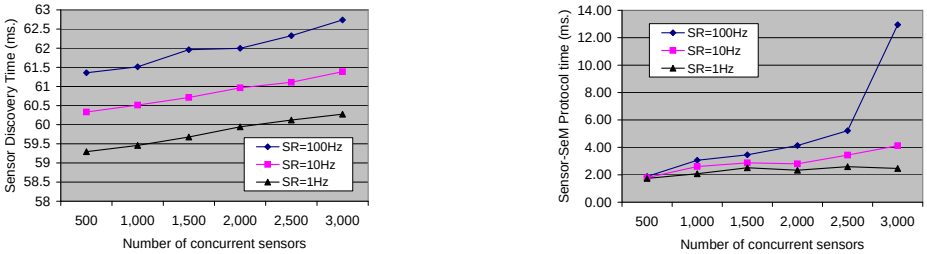



**Fig. 6.** SeM's Adaptability in Dynamic Environment (average)

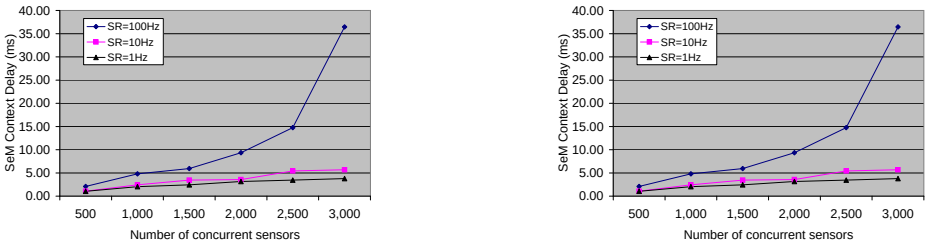
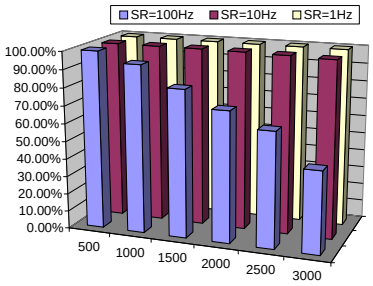



**Fig. 7.** SeM's Effectiveness for Context Delivery

Fig. 7 reports on context data delivery delay at SeMs and context consumers. Each value is computed as the average of at least 600,000 samples. Note that some delay $\Delta T$ does not mean that a SeM's processing capacity for context data is 1 per $\Delta T$, because SeM is implemented using parallel processing. In other words, a SeM has multiple dispatching threads running concurrently, and macroscopically speaking many context data are being handled during $\Delta T$. It is intuitively understood that the larger the workload of the GlobeCon run time

environment, the longer the context delivery delay. It is worth mentioning that even though the growth is linear with respect to network size, the growing rate for SR=100Hz is larger than the other two SR, and it becomes even larger when the number of concurrent sensors exceeds 2500. This is because the SeM is reaching its processing capacity around that point, various SeM's queues are about full (i.e., context data has prolonged enqueue time) and the SeM starts to drop more context data while new ones arrive. However, the delay of 40ms at SeMs and 80ms at LoCoMs under extremely high workload (i.e., 300,000 messages per second on a single SeM) is still very good, and this delay is maintainable through the SeM's lifetime (30 minutes of our simulation runs).




| #. of sensors | SR=100Hz | SR=10Hz | SR=1Hz |
|---|---|---|---|
| 500 | 50/50 | 5/5 | 0.5/0.5 |
| 1000 | 94/100 | 10 /10 | 1/1 |
| 1500 | 124/150 | 15/15 | 1.5/1.5 |
| 2000 | 146/200 | 19.9/20 | 2/2 |
| 2500 | 161/250 | 24.9/25 | 2.5/2.5 |
| 3000 | 138/300 | 29.7/30 | 3/3 |

SeM's capacity for processing context items: #. of processed vs. #. of received (k/sec.)

**Fig. 8.** SeM's SeM throughput

Fig. 8 gives the results for SeM's throughput in term of percentage and number of processed context items. Fig. 8 (left) shows that SeM's throughput starts to drop dramatically when SR=100Hz and the number of sensors is 2500. Under the same setting, the SeM yield its maximum processing capacity, which is around 161K items per second as shown in Fig. 8 (right). This performance depends on the speed of underlying communication networks and the horse power of SeM's hosting machine. According to results for the setting of 1000-3000 sensors and SR=100Hz in Fig. 8 (right), it is reasonable to draw the conjecture on the throughput for SR=10Hz and SR=1Hz with two possible cases: (a) if the incoming workload exceeds the system's potential capacity, the throughput for SR=10Hz and SR=1Hz should be at least as good as that for SR=100Hz; (b) if the incoming workload is less than the system capacity (where the throughput is impossible to exceed the incoming workload) the throughput for SR=10Hz and SR=1Hz should be 100% (i.e., the drop rate is zero). However the results show that this conjecture would not correct (i.e. cases where SR=10Hz). Therefore there must be other system parameters than SR and number of sensors that affect the system performance. To gain an insight into this behavior we run another set of simulations to evaluate how queue size and number of SeM's dispatching threads affect the performance in terms of SeM's throughput.

*Effect of queue size and number of dispatching threads* We only present a subset of our experimental results, which are most illustrative to the effects of the two parameters. Generally, smaller queue size causes higher drop rate, while larger queue size produces longer context enqueue time especially when a SeM is operating in near maximum capacity. Our results show that when SR=1HZ,

both SeM's throughput and context delay are not sensitive to queue size as long as it is greater than 10. When SR exceeds 10Hz, the effect of queue size on SeM's throughput becomes apparent.Fig. 9 (left) gives the SeM's throughput under different queue size with sensors' sampling rate set to 10Hz. From Fig. 9 (right), SeM's throughput is improved while we increase the number of dispatching threads from 200 to 400. However, it becomes worse while increasing the number to 500. This is because the overhead of parallelism (i.e., context switching, kernel crossing and mutex operations) offsets the benefit of parallelism near this point.

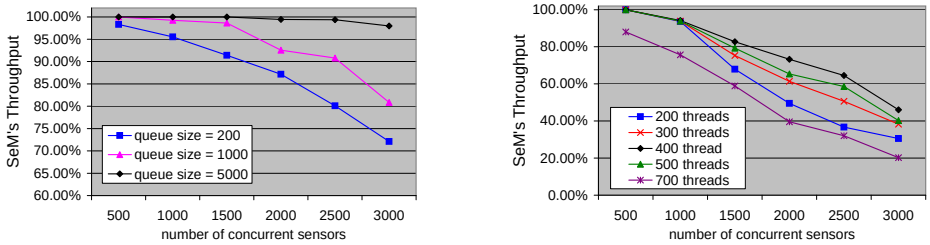



**Fig. 9.** Effect of queue size and number of dispatching threads

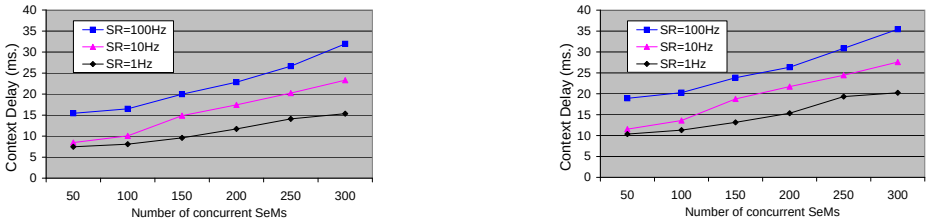



**Fig. 10.** LoCoM's Effectiveness for Context Delivery

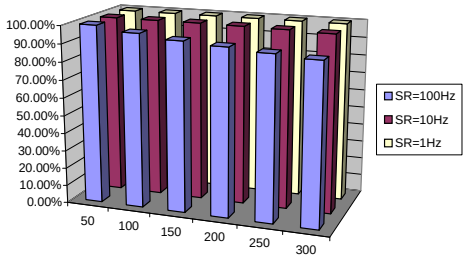



**Fig. 11.** LoCoM's throughput

**Performance of LoCoM** We conducted experiments on LoCoM as well to study a LoCoM's performance under various workloads. In this set of experiments, the number of LoCoM is set to one, and the number of context consumer is set to 1000, while the number of SeM and sensors' sampling rates are varied. All the SeMs are configured with incoming queue sizes equal to 5000, outgoing queue sizes equal to 1000 and number of dispatching threads equal to 400. Each

SeM has an average of 10 registered context sensors. All participating consumers subscribe context information from LoCoM only.

The results of LoCoM's context delivery delay are shown in Fig. 10, which demonstrates the advantage of our core design idea - hierarchical context aggregation. In fact, the hierarchical structure implicitly balances the workload among multiple distributed SeMs and LoCoMs. If we compare the workload at SeM and LoCoM in terms of context arrival rate, a LoCoM with 300 concurrent SeMs is actually exposed to the same amount of workload as a SeM with 3000 concurrent sensors. However, a LoCoM is able to achieve shorter context delivery latency than a SeM having the same amount workload. LoCoM's throughput under various sampling rates and different numbers of connected SeMs is shown in Fig. 11, which again demonstrates the advantage of our design (i.e., a LoCoM outperforms a SeM of the same amount workload in terms of throughput).

Our results on LoCoM's maintenance overhead, and adaptability follow similar trends as those of SeM's, so the statistics are omitted due to page limit.

## 5 Conclusions

In this paper, we studied Globecon from an implementation and experimental point of view. We believe our measurement studies are of essential practical importance, because they provide guidance for the design of real-life systems, help to pinpoint potential bottlenecks and give insight on approaches for improvements. We applied four groups of evaluation metrics to gain insights, from different points of view, on how GlobeCon behaves under various configurations and workloads ranging from normal to heavy. Our results show that both SeM and LoCoM scale well while increasing the network size: the context delay is only tens of milliseconds even under the workload of 300,000 msg/sec; SeM achieved 100% thoughtput under the workload of 15,000 msg/sec with 1500 concurrent sensors and 1000 concurrent consumers; LoCoM achieved 90% throughput under workload of 300,000 msg/sec. Under the extreme workloads that exceeds 250,000 msg/sec, a SeM tends to produce a higher growing-rate on context delivery latency while increasing the number of its concurrent sensors, however, the SeM is able to selectively[4] adjust the sensors' sampling rate dynamically and tune its run-time performance on-the-fly. To sum up, GlobeCon is demonstrated to be a practical and scalable context management system that is able to efficiently manage context producers/consumers in highly dynamic environments with little maintemence overhead.

## References

1. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks 3, (1997) 421-433.
2. Apache jUDDI. http://ws.apache.org/juddi/, 2008.

[4] Selection based on pre-defined ranking, e.g., contexts with higher quality requirements keep higher sampling rate.

3. Buchholz, T., Linnhoff-Popien, C.: Towards realizing global scalability in context-Aware systems. In 1st Int'l Workshop on Location- and Context-Awareness, 2005.
4. Chen, H., Finin, T., Joshi, A.: Semantic web in the context broker architecture. In Proc. of IEEE Int'l Conf. on Pervasive Computing and Communications, 2004.
5. Chen, G., Kotz, D.: Solar: An open platform for context-aware mobile applications. In Proc. of 1st Int'l Conf. on Pervasive Computing, 2002. pp.41-47.
6. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Dartmouth College Technical Report TR2000-381, November 2000.
7. Cheverst, K., Davies, N., Mitchell K., Friday, A.: Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In Proc. of 6th Annual Int'l Conf. on Mobile Computing and Networking, 2000. pp.20-31.
8. Clement, L., Hately, A., Riegen, von C., Rogers, T.: UDDI Version 3.0.2. UDDI Spec Technical Committee Draft, October 2008.
9. Dey, A. K., Salber, D., Abowd, G. G.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Journal of Human-Computer Interaction (HCI) 16(2-4). (2001) 97-166.
10. Fahy, P. and Clarke, S.: CASS: a middleware for mobile context-aware applications. In Workshop on Context Awareness, MobiSys, 2004
11. Gervais, E., Liu, H., Nussbaum, D., Roh, Y.-S., Sack, J.-R., Yi, J.: Intelligent Map Agents - A ubiquitous personalized GIS. Accepted by ISPRS Journal of Photogrammetry and Remote Sensing, special issue on Distributed Geoinformatics, 2007.
12. Gervais, E., Nussbaum, D., Sack, J.-R.: DynaMap: a context aware dynamic map application. presented at GISPlanet, Estoril, Lisbon, Portugal, May 2005
13. Glassey, R., Stevenson, G., Richmond, M., Nixon, P., Terzis, S., Wang, F., Ferguson, R. I.: Towards a middleware for generalised context management. In 1st Int'l Workshop on Middleware for Pervasive and Ad Hoc Computing, 2003.
14. Gu, T., Pung, H. K., Zhang, D. Q.: A service-oriented middleware for building context aware services. Network and Computer Applications 28(1). (2005) 1-18
15. Gu, T., Qian, H. C., Yao, J. K., Pung, H. K.: An architecture for flexible service discovery in OCTOPUS. In Proc. of the 12th Int'l Conference on Computer Communications and Networks (ICCCN), Dallas, Texas, October 2003.
16. JADE: Java Agent DEvelopment Framework. http://jade.tilab.com/, 2008.
17. Keeping Canada on Time. http://www.nrc-cnrc.gc.ca/eng/education/innovations /discoveries/clock.html, 2008.
18. Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., Malm, K. J.: Managing context information in mobile devices. IEEE Pervasive Computing, 2(3). (2003) 42-51.
19. Lu, K., Nussbaum, D., Sack, J.-R.: GlobeCon - A Scalable Framework for Context Aware Computing. In Proc. of 2nd European Conf. on Smart Sensing and Context, 2007. pp.190-206.
20. Mitchell, K.: A survey of context-aware computing, Lancaster University, Technical Report, March 2002.
21. Satyanarayanan, M.: Pervasive computing: vision and challenges. IEEE Personal Communication Magazine, 8(4). (2001) 110-17
22. Schilit, B. and Theimer, M.: Disseminating active map information to mobile hosts. IEEE Network. 8(5). (1994) 22-32.
23. SUN-Web Services Developer Pack. http://java.sun.com/webservices/jwsdp, 2008.
24. Strang, T., Claudia L., P.: A context modeling survey. In UbiComp 1st Int'l Workshop on Advanced Context Modeling, Reasoning and Management, (2004) 34-41.
25. Tanenbaum, A. S.: Computer Networks. Prentice Hall, Inc. 3rd Edition, 1996.
26. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems, 10(1). (1992) 91-102.