A Framework of Sensor - Cloud Integration Opportunities and Challenges

Mohammad Mehedi Hassan
Dept. of Computer Engineering
Kyung Hee University
1 Seocheon, Giheung-gu, Yongin
Gyeonggi-do, 446-701, South Korea
hassan@khu.ac.kr

Biao Song
Dept. of Computer Engineering
Kyung Hee University
1 Seocheon, Giheung-gu, Yongin
Gyeonggi-do, 446-701, South Korea
bsong@khu.ac.kr

Eui-Nam Huh
Dept. of Computer Engineering
Kyung Hee University
1 Seocheon, Giheung-gu, Yongin
Gyeonggi-do, 446-701, South Korea
johnhuh@khu.ac.kr

ABSTRACT
In the past few years, wireless sensor networks (WSNs) have been gaining increasing attention because of their potential of enabling of novel and attractive solutions in areas such as industrial automation, environmental monitoring, transportation business, health-care etc. If we add this collection of sensor derived data to various Web-based social networks or virtual communities, blogs etc., we can have a remarkable transformation in our ability to "see" ourselves and our planet. Our primary goal is to facilitate connecting sensors, people and software objects to build community-centric sensing applications. However, the computational tools needed to launch this exploration may be more appropriately built from the data center "Cloud" computing model than the traditional HPC approaches. In this paper, we propose a framework to enable this exploration by integrating sensor networks to the emerging data center "cloud" model of computing. But there are many challenges to enable this framework. We propose a pub-sub based model which simplifies the integration of sensor networks with cloud based community-centric applications. Also there is a need for internetworking cloud providers in case of violation of service level agreement with users. We discussed these issues and proposed reasonable solutions.

Categories and Subject Descriptors
C.2.4 [**Distributed Architecture**]: Distributed Applications

General Terms
Design, Economics and Experimentation

Keywords
Sensor network, Cloud computing, pub/sub, dynamic collaboration, event matching algorithm

ICUIMC-09, January 15-16, 2009, Suwon, S. Korea

1. INTRODUCTION
In recent years, Cloud computing [1] emerges as a new computing paradigm to provide reliable services delivered through next-generation data centers that are built on virtualization technologies such as VMs, dynamically "provisioned" on demand as a personalized resource collection to meet a specific service - level agreement, which is established through a "negotiation" and accessible as a composable service via "Web 2.0" technologies [2]. Consumers will be able to access applications and data from a "Cloud" anywhere in the world on demand. In other words, the role of the cloud is to provide a place where application "suppliers" can make apps available to clients. The applications are then hosted as "Software as a Service (SaaS)" and the Cloud automatically scale to meet client demand.

However, there is another revolution going on driven by an explosion in our ability to create, store and mine digital information gathered not only by scientific simulations but also by the vast sea of sensors we have connected to the global network. In the past few years, wireless sensor networks (WSNs) have been gaining increasing attention, both from commercial and technical point of views, because of their potential of enabling of novel and attractive solutions in areas such as industrial automation, asset management, environmental monitoring, transportation business, health-care etc. If we add this collection of sensor derived data to various Web-based social networks or virtual communities, blogs, and the general wiki-corpus etc., we can have a remarkable transformation in our ability to "see" ourselves and our planet. Our primary goal is to facilitate connecting sensors, people and software objects to build community-centric sensing applications where people can share and analyze real time sensor data. For example, a virtual community of doctors for monitoring patient healthcare for virus infection using various sensors, portal for sharing real time traffic information, real-time environment data monitoring and analysis etc. To enable this exploration, sensor data of all types will drive a need for an increasing capability to do analysis and mining on-the-fly. However, the computational tools needed to launch this exploration may be more appropriately built from the data center "cloud" computing model than the traditional HPC approaches or Grid approaches that have worked so well for supercomputing user.

Generally supercomputers are designed for extreme capability on floating point intensive applications. The "business model" for

supercomputers is based on optimizing peak floating-point performance for individual applications per megawatt of energy; purchase price and staff time. Cloud data centers are designed to provide on-demand capacity for the application providers. While the supercomputers are typically operates on a two party contracts Cloud data centers often involve three parties: the center, the application provider and the application user. The application providers want to be sure that the user never 'sees' the data center, only the application service. The business model here is closer to optimizing user experience per megawatt hour while also making the application development experience as simple as possible [2, 4] .

So in this paper, we propose a framework of Sensor-Cloud connection to utilize the ever-expanding sensor data for various next generation community-centric sensing applications as SaaS on the Cloud that can have vast scientific values as well as a huge economic value. But there are many challenges exist to enable this framework in terms of processing and disseminating data to user as the entire network is dynamic. We propose a content-based pub-sub model which simplifies the integration of sensor network with cloud based community centric applications. Also to deliver published sensor data or events to appropriate users of Cloud applications who subscribed, it is needed to match published events with subscriptions efficiently. So we also present a fast and scalable event matching algorithm called *Statistical Group Index Matching (SGIM)* suitable to our application scenario and its performance is evaluated with existing algorithms in the literature in a Cloud based ubiquitous health-care application.

Furthermore, as consumers of different SaaS applications rely on Cloud providers (CLP) to supply all their computing needs (process, store and analyze huge sensor data and user generated data) on demand, they will require specific QoS to be maintained by their providers in order to meet their objectives and sustain their operations. Existing commercial Cloud services are proprietary in nature. They are owned and operated by individual companies. Each of them has created its own closed network, which is expensive to setup and maintain. So, if the CLP is unable to provide quality of service to the end-user requests (in the case when huge sensor data needs to process for critical health care scenario, earthquakes monitoring etc), it may result in service – level agreement (SLA) violation and end up costing the provider. Our framework also addresses this issue and enables a VO based dynamic collaboration between other CLP providers.

The paper is organized as follows: Section 2 describes our application scenario. Section 3 presents related works. Section 4 describes sensor-cloud framework and VO based dynamic collaboration architecture. Section 5 presents the proposed SGIM algorithm. Section 6 shows the performance results of the algorithm compare to existing ones and finally Section 7 concludes the paper.

2. AN APPLICATION SCENERIO

A virtual community consisting of team of researchers have come together to solve a complex problem and need data storage, compute capability, security; and they need it all provided now. For example, this team is working on an outbreak of a new virus strain moving through a population. This requires more than a Wiki or other social organization tool. They deploy bio-sensors on patient body to monitor patient condition continuously and to use this data for large multi-scale simulations to track the spread of infection as well as the virus mutation and possible cures. This may require computational resources and a platform for sharing data and results that are not immediately available to the team. Traditional HPC approach like Sensor-Grid [6] model can be used in this case, but setting up the infrastructure to deploy it so that it can scale is not easy in this environment. However, Amazon EC2 like Cloud data centre shows us that many copies in a data center will work. But current Cloud providers did not address the issue of integrating sensor network with Cloud applications and thus have no infrastructure to support this scenario. The VO needs a "place" that can be rapidly deployed with social networking and collaboration tools (including security), other specialized applications and tools that can compose sensor data and disseminate them to the VO users based on their subscriptions. Here, the researchers need to register their interests to get various patients' state (blood pressure, temperature, pulse rate etc.) from bio-sensors for large scale parallel analysis and to share this information with each other to find useful solution to the problem. So the sensor data needs to aggregate, process and disseminate based on subscriptions.

On the other hand, as huge sensor data require computational power and storage, the Cloud provider may not handle this for various sensor data related applications. They may require dynamic collaboration with other Cloud providers for burst resource demand. Our framework addresses above issues and provides solutions.

3. RELATED WORK

To the best of our knowledge, the issue of integrating sensor network to Cloud computing model is not yet addressed in the literature. Sensor-Grid [3] architecture is already there but as we mentioned earlier it is not suitable in the current application scenarios. In [4], it is also discussed that current Grid computing applications are different from the Cloud applications. Grid is generally focus on HPC related application while Cloud focuses on general purpose applications. Users are now moving from HPC workloads to more generic workloads for which Cloud is more appropriate solution.

To integrate sensor networks to Cloud, we propose a content-based pub-sub model. A Pub/Sub system [5] encapsulates sensor data into events and provides the services of event publications and subscriptions for asynchronous data exchange among the system entities. Several Pub/Sub systems have been developed in recent years in the context of WSNs most notably in [6]-[9].

Mires [6] is a publish/subscribe architecture for WSNs. Basically sensors only publish readings if the user has subscribed to the specific sensor reading. Messages can be aggregated in cluster heads. Subscriptions are issued from the sink node (typically directly connected to a PC), which then receives all publications.

TinySIP [7] supports session semantics, publish/subscribe, and instant messaging. It offers support for multiple gateways. Most communication is done by addressing individual devices. As device addresses are related to the gateway being used, changing the gateway on the fly is difficult.

DV/DRP [8] is another publish/subscribe architecture for WSNs. DV/DRP stands for Distance Vector/Dynamic Receiver Partitioning. Subscriptions are made based on the content of the desired messages. Subscriptions are flooded in the network. Intermediate nodes aggregate subscriptions. They forward publications only if there is an interest for this publication. Because of the complexity of matching subscriptions to arbitrary data packets it would be difficult to implement this protocol on the devices we target.

MQTT-S [9] is an open topic-based pub-sub protocol that hides the topology of the sensor network and allows data to be delivered based on interests rather than individual device addresses. It allows a transparent data exchange between WSNs and traditional networks and even between different WSNs.

In our framework, like MQTT-S, all of the system complexities reside on the broker's side but it differs from MQTT-S in that it uses content-based pub-sub broker rather than topic-based which is suitable for our application scenarios. When an event is published, it is transmitted from a publisher to one or more subscribers without the publisher having to address the message to any specific subscriber. Matching is done by the pub-sub broker outside of the WSN environment. In content-based pub-sub system, sensor data has to be augmented with meta-data to identify the different data fields. For example, a meta-data of a sensor value (also event) can be body temperature, blood pressure etc.

To deliver published sensor data or events to subscribers, an efficient and scalable event matching algorithm is required by the pub-sub broker. Several event matching algorithms are proposed in the literature most notably in [10]-[12].

In [10], authors proposed two event matching algorithms - sequential and sub-order. In sequential algorithm, according to each predicate in event, searching space is gradually reduced by deleting subscriptions which are not satisfied. The advantage is that no pre-processing for subscriptions and no additional index is required. When the number of attribute increases, the cost approaches to a constant number. But the drawback is that at least we need to check every predicate once in the subscriptions. That means the cost is no less than the total no. predicates. The second algorithm, sub-order, reduces the expected number of predicate evaluations by analyzing the expected cost differences when subscriptions are evaluated in different orders. In case of sub-order, connect the same predicates together by using an index called "linked chain" that can help to find same predicates quickly, and then it is possible to quickly reserve or delete predicates. The advantage is that by pre-processing, it can avoid great amount of comparison during the searching and also it can further reduce the searching cost when there are many same predicates in the subscriptions. But the disadvantage is that it is hard or even impossible to make chain sometimes, for instance in range predicate case. Another problem is that here every two predicates, if they are same, share a chain. Thus it needs to maintain a complete graph and bring so much overload when inserting and deleting subscriptions.

Forwarding algorithm in [11] matches event data to subscriptions that include a single predicate. It groups predicates using attributes and operators, and indexes the values of predicates. The advantage is that searching is very fast. But when predicates are in range or highly overlapped, this algorithm results in separate groups of predicates by matching operators and finds correct subscriptions by combining the results of different groups.

In VCR Indexing described in [12], range of each subscription are transferred into many overlapping rectangles and when event data comes, rectangles which contain this event point can be found and then subscriptions are found which contain those rectangles. The advantage is that it performs better when the subscriptions are highly-overlapping. But its disadvantage is that it is not scalable. If the number of attribute increases the overhead of rectangle construction and storage increases dramatically.

Our event matching algorithm targets range predicate case suitable to our application scenario and it is also efficient and scalable when the number of predicates increases.

In [2], authors presented a vision for the creation of global Cloud exchange for trading services. But authors did not mention about dynamic collaboration between cloud providers.

In [14], authors proposed a VO based CDN peering framework. We were inspired by this work and design our framework to support VO based dynamic collaboration. .

4. PROPOSED FRAMEWORK

4.1 System Overview

The framework is shown in figure 1. The terminologies used to describe the system architecture are listed in Table 1. In this framework, sensor data are coming through gateways to a pub/sub broker. Pub sub broker is required in our system to deliver information to the consumers of SaaS applications as the entire

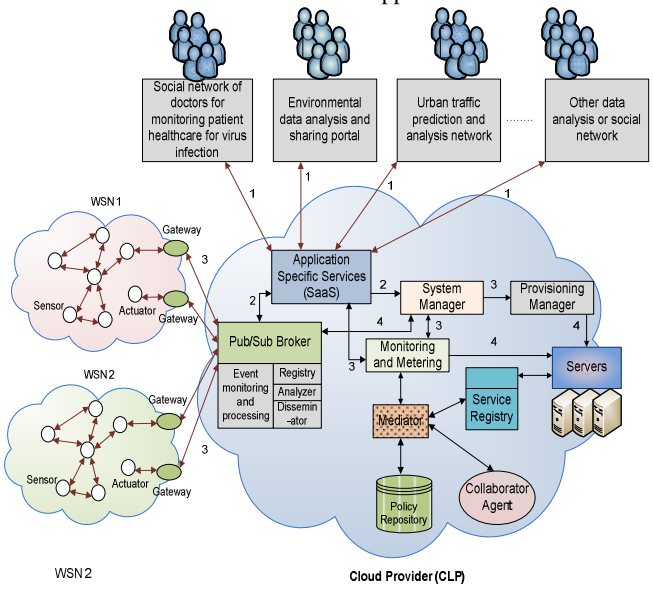

Figure 1: A framework of Sensor – Cloud Integration

network is very dynamic. On the WSN side, sensor or actuator (SA) devices may change their network addresses at any time.

Wireless links are quite likely to fail. Furthermore, SA nodes could also fail at any time, and rather than being repaired, it is expected that they will be replaced by new ones.

Besides, different SaaS applications can be hosted and run on any machines anywhere on the cloud. In such situations, the conventional approach of using *network address* as communication means between the SA devices and the applications may be very problematic because of their dynamic and temporal nature. Moreover, several SaaS applications may have an interest in the same sensor data but for different purposes. In this case, the SA nodes would need to manage and maintain communication means with multiple applications in parallel. This might exceed the limited capabilities of the simple and low-cost SA devices. So pub-sub broker is needed and it is located in the Cloud side because of its higher performance in terms of bandwidth and capabilities. It has four components describes as follows:

Table 1: List of commonly used terms

Terminology	Description
Gateway	Sensor data comes to the system through gateway
Pub/Sub Broker	Responsible to monitor, process and deliver events to users (registered) through SaaS applications
SaaS Application	Software as a Service application which combines data and application together and runs on the Cloud server
System Manager (SM)	Manages the computer resources
Provisioning Manager (PM)	Carve out the systems from the cloud to deliver on the requested service. It may also deploy the required images.
Monitoring and Metering (MM)	Tracks the usage of the primary cloud resources as well as the resources of collaborator CLP's so the resources used can be attributed to a certain user.
Service Registry (SR)	Discovers and stores resource and policy information to local domain
Mediator	Responsible for negotiation among CLPs and management of operations within a VO
Policy Repository (PR)	A storage of Web server, mediator and VO policies
Collaborator Agent (CA)	A resource discovery module in the collaborating CLPs environment

Stream monitoring and processing component (SMPC): The sensor stream comes in many different forms. In some cases it is raw data that must be captured, filtered and analyzed on the fly and in other cases it is stored or cached. The style of computation required depends on the nature of the streams. So the SMPC component running on the cloud monitors the event streams and invokes correct analysis method. Depending on the data rates and the amount of processing that is required, SMP manages parallel execution framework on Cloud.

Registry component (RC): Different SaaS applications register to pub-sub broker for various sensor data required by the community user. For each application, registry component stores user subscriptions of that application and sensor data types (temperature, light, pressure etc.) the application is interested in. Also it sends all user subscriptions along with application id to the disseminator component for event delivery.

Analyzer component (AC): When sensor data or events come to the pub-sub broker, analyzer component determines which applications they are belongs to and whether they need periodic or emergency deliver. The events are then passed to the disseminator component to deliver to appropriate users through SasS applications.

Disseminator component (DC): For each SaaS application, it disseminates sensor data or events to subscribed users using an event matching algorithm (described in section 6). It can utilize Cloud's parallel execution framework for fast event delivery.

The pub-sub components workflow in our framework is as follows:

① Users register their information and subscriptions to various SaaS applications which then transfer all this information to pub/sub broker registry.

② When sensor data reaches to the system from gateways, event/stream monitoring and processing component (SMPC) in the pub/sub broker determines whether it needs processing or just store for periodic send or for immediate delivery.

③ If sensor data needs periodic/ emergency delivery, the analyzer determines which SaaS applications the events belong to and then passes the events to the disseminator along with application ids.

④ The disseminator, using event matching algorithm, finds appropriate subscribers for each application and delivers the events for use.

4.2 VO Based Dynamic Collaboration

Besides the pub-sub broker, we propose to include three other components like [14]- mediator, policy repository (PR) and collaborator agent (CA) along with system manager, provisioning manager, monitoring and metering and service registry (which are normally present in a Cloud model) in our Sensor-Cloud framework to enable VO based dynamic collaboration of primary Cloud providers with other Cloud providers in case of SLA violations for burst resource demand. These three components collectively act as a "gateway" for a given CLP in creation of a new VO. They are described as follows:

Mediator: The (resource) mediator is a policy-driven entity within a VO to ensure that the participating entities are able to adapt to changing circumstances and are able to achieve their objectives in

a dynamic and uncertain environment. Once a VO is established, the mediator controls which resources to be used of the collaborating CLPs, how this decision is taken, and which policies are being used. When performing automated collaboration, the mediator will also direct any decision making during negotiations, policy management, and scheduling. A mediator holds the initial policies for VO creation and works in conjunction with its local Collaborating Agent (CA) to discover external resources and to negotiate with other CLPs.

Policy Repository (PR): The PR virtualizes all of the policies within the VO. It includes the mediator policies, VO creation policies along with any policies for resources delegated to the VO as a result of a collaborating arrangement. These policies form a set of rules to administer, manage, and control access to VO resources. They provide a way to manage the components in the face of complex technologies.

Collaborating Agent (CA): The CA is a policy-driven resource discovery module for VO creation and is used as a conduit by the mediator to exchange policy and resource information with other CLPs t. It is used by a primary CLP to discover the collaborating CLPs' (external) resources, as well as to let them know about the local policies and service requirements prior to commencement of the actual negotiation by the mediator.

A VO may vary in terms of purpose, scope, size, and duration. Hence, VOs are of two types: short-term on-demand VOs and long-term VOs with established SLAs. In a long-term VO, CLPs collaborate for longer period of time and such a VO remains for the duration of the event. In this situation we would expect negotiation to include a human-directed agent to ensure that any resulting decisions comply with participating companies' strategic goals. In order to maintain the SLA to meet its QoS obligations, we propose that time-critical agreements for a short-term VO should be automatically negotiated. The VO formation scenario is explained in figure 2. In figure 3, the VO creation steps are shown which are explained as follows –

Step 1: A (primary) CLP provider realizes that it cannot handle a part of the workload on its Web server(s). A VO initialization request is sent to the mediator.

Step 2 and Step 3: The mediator instance obtains the resource and access information from the SR

Step 4: The mediator instance on the primary CLP's behalf generates its service requirements based on the current circumstance and SLA requirements of its customer(s) and passes the service requirements to the local Collaborating Agent (CA).

Step 5 and Step 6: If there is any preexisting collaborating arrangements (for a long term scenario) then these will be returned at this point. Otherwise, it carries out short term negotiations with the CAs of other CLPs.

Step 7: SLAs and policies of other CAs are sent to the mediator instance.

Step 8 and Step 9: The mediator instance consults with primary policy repository and find out the best CLP collaborators.

Step 10: The mediator instance then requests for a new VO creation to primary CA.

Step 11: When primary CA acquires resources from other collaborating CLPs that meet its SLA with the customers, the new VO becomes operational. If no CLP is interested in such peering, VO creation through re-negotiation is resumed from Step 4 with reconsidered service requirements.

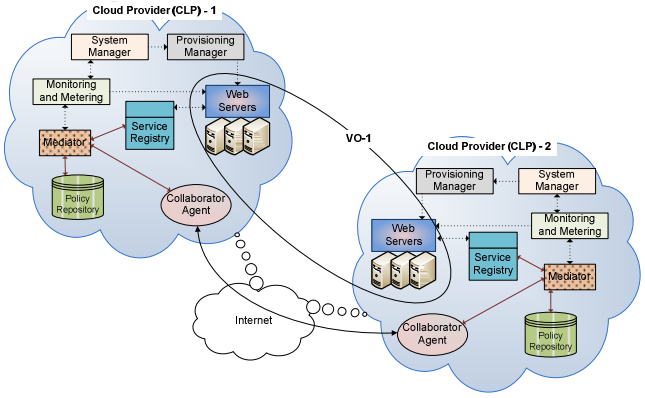

Figure 2: An architecture assisting dynamic VO formation between CLPs

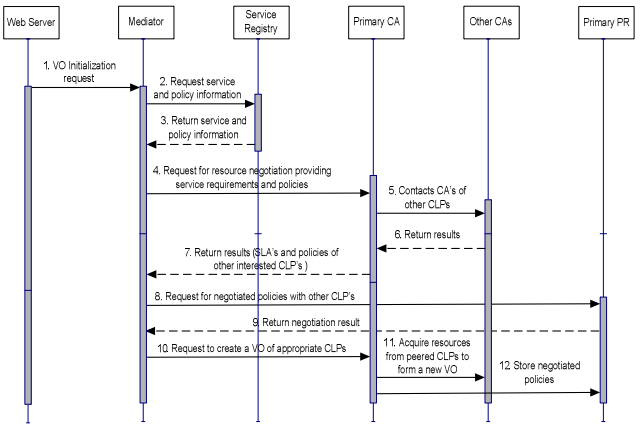

Figure 3: Sequence diagram of formation of a new VO

But there are serious issues involved in such agreements including trust and the potential commercial sensitivity of information about the current state and costs of a CLP. One solution to these problems is to utilize a cryptographically secure auction [13], which hides both the valuations that CLPs place on their resources and who is participating in the auction.

5. EVENT MATCHING ALGORITHM

5.1 The Event Matching Problem

To deliver publish events to appropriate consumers who subscribed an efficient event matching algorithm is required for our Sensor-Cloud framework. We are targeting the Cloud-based healthcare monitoring system (described earlier) where doctors and researchers express their interests into a range (i.e. $35 <$ body temperature < 37). Formally, in our system, a subscription S is expressed by a pair (Sid, P) where Sid is the subscriber's ID (subscriber ID for short) and P is a set of predicates specifying subscriber's interest. P consists of five elements as shown below-

$P = \{event\_type, N, op, mPV, MPV\}$ where
$event\_type =$ detail event name (i.e. body temperature, blood pressure etc.)
$mPV =$ minimum threshold value of predicate
$MPV =$ maximum threshold value of predicate
$N =$ count of predicates (i.e. '1' for single event data matching and '2' for two event data matching with mPV and MPV at the same time)
$op =$ represents operators like "><" *or* "< || >" *or* "< && >".
Here is one example of a subscription and an event in the system Subscription S: [body temperature, 1, ><, 29, 32] which contains two predicates that are joined together to specify a range value or range predicate i.e. $p_1 =$ body temperature > 29 and $p_2 =$ body temperature < 32 . We can also express it as $29 <$ body temperature < 32. Event e: [body temperature = 30] . An event satisfies a subscription only if it satisfies all predicates in the subscription. Here, the event e = [body temperature =30] satisfies the subscription *S* since body temperature is within the range. So the matching problem is: Given an event e and a set of predicates in subscription set *S*. We need to find all subscriptions in set *S* that are satisfied by *e*. Our pub-sub broker supports various expressions of predicates. First, "$(mPV < data < MPV)$" is used when consumers want to know normal patterns of sensed data. Second, "$(mPV > data \,||\, MPV < data)$" is used when consumers need to receive unusual states of the patient and third, "$(mPV > data_1 \,\&\&\, MPV < data_2)$" is used to support for couple of sensor data observed at the same time such as blood pressure.

5.2 Proposed Algorithm

We now describe the Statistical Group Index Matching (SGIM) Algorithm. SGIM algorithm operates in two stages. In first stage it preprocesses subscriptions by grouping them by the predicates corresponding to the relevant properties of events. The basic grouping idea from statistics is employed to decide the number of groups. In the second stage matching subscriptions or predicates are derived sequentially. All predicates stored in the system are associated with a unique ID. Similarly subscriptions are identified with subscription id.

Let S is a set of subscriptions, $S = \{s_1, s_2, ...s_{n-1}, s_n\}$ where n is the total no. of subscriptions and P is a set of predicates in S, $P = \{p_1, p_2 ....p_{n-1}, p_m\}$ where m is the total no. of predicates in a subscription. In our system, we have two predicates in a subscription (i.e. $data > mPV$ and $data < MPV$) and these two predicates is used to group the subscriptions. We define a set S' that contains all the subscriptions of S sorted by mPV value in ascending order. Then we define a grouping sequence $(mG_1, mG_2, ...mG_g)$ such that $g = \lfloor 1 + \log_2(n) \rfloor$ and $\sum_{r=1}^{g} mG_r = S'$, where n is the total no. of subscriptions in S'. The grouping space, denoted by $SP(S', g)$, is defined as the set containing all such group sequences over S' and g . Now each $mG_{i=1...g} \in SP(S', g)$ contains $k = \lceil n / g \rceil$ no. of subscriptions, So group index (a range value), $gI_i = (_{\min} mPV - _{\max} mPV)$ is created for each $gI_i \in mG_{i=1...g}$. Here our grouping sequence is called almost balanced grouping sequence since every group contains same no. of subscriptions except the last one which may or may not contain the same no. of subscriptions. It depends on the value of g and n .

Again from each $mG_{i=1...g} \in SP(S', g)$, we define another grouping sequence $(MG_1, MG_2, ...MG_h)$ which are sorted by MPV value in ascending order such that $h = \lfloor 1 + \log_2(k) \rfloor$ and $\sum_{t=1}^{h} MG_t = mG_i$, for any $mG_{i=1..g} \in SP(S', g)$. Then each $MG_{l=1..h} \in SP(mG_{i=1..g}, h)$ contains $q = \lceil k / h \rceil$ no of subscriptions and thus group index $hI_i = (_{\min} MPV - _{\max} MPV)$ is created for each $hI_i \in MG_{i=1...h}$.

After grouping of subscriptions in the above way, first predicate of an event is compared with group index $gI_1 \in mG_1$ and if any match found then second predicate is compared with group indexes $hI_i \in MG_{i=1...h}$ within $MG_{i=1..h} \in SP(mG_1, h)$ and so on. In this way all groups are found that matches with event data. Finally sequential matching is done in the selected groups to find the subscriptions that are satisfied by all predicates in the event. The pseudo code of grouping and event method is shown in figure 4 and figure 5.

5.2.1 Analysis of the SGIM Algorithm

SGIM algorithm looks very simple. However, in case of normal range predicates or highly overlapping range predicates, it can greatly reduce the search space for fast event matching since it can directly choose or delete one or more group of subscriptions just by checking the index of groups. Then using sequential searching, when any predicate in a subscription is evaluated to false, there is no need to evaluate the remaining predicates. Furthermore, if this predicate is shared by many subscriptions, all remaining predicates in these subscriptions can be ignored, which can significantly reduce the evaluation overhead. The evaluation cost of SGIM is linear to the number of subscriptions.

The algorithm for inserting a new subscription in the system is very similar to the event matching algorithm. First the algorithm finds the group index which matches the subscription and insert into that group. If necessary, the group index may be updated for that subscription insertion. The cost of the inserting algorithm is close to the event matching cost. Deletion algorithm is very easy. Just we need to find the exact group index of the subscriptions to be deleted and then delete that from the group and if necessary group index is updated. If whole group needs to be deleted then we can search that group index and delete it.

6. EVALUATION

In this section we present our experimental methodology and simulation results for our SGIM algorithm. We ran all experiments on a windows machine with Pentium Celeron 1.6 GHz CPU and 1 GB memory. We compare our algorithm with sequential sub-order, forwarding and naive algorithms. Naïve, a baseline algorithm, evaluates the subscriptions independently. Specifically, it evaluates the subscriptions one by one and, for

each subscription, evaluates its predicates until either one of them becomes *false* or all of them are evaluated. We implement all the algorithms along with SGIM in Visual C++.

Input:
1. A set of subscriptions $S = \{s_1, s_2 .... s_{n-1}, s_n\}$ where n indicates the total number of subscriptions
2. A set of predicates P in S , $P = \{p_1, p_2 .... p_{m-1}, p_m\}$ where m indicates the number of predicates in a subscription. In our case, m=2
3. A global variable j initially equals to 1. p_j indicates no. of predicates used for grouping.

Body:
4. Procedure Grouping (S, p_j, n) ▷ grouping S by using predicate p_j
5. $S' = Sort(S, p_j)$ ▷ sorting S in ascending order by predicate p_j value and make new index S'
6. $g = \lfloor 1 + \log_2(n) \rfloor$ ▷ g is the theoretical number of groups
7. $k = \lceil n / g \rceil$ ▷ k is the number of subscriptions in each group
8. $count = 0$ ▷ count the number of subscriptions in each group
9. $S\_temp = \phi$ ▷ S_temp temporarily stores the subscription group
10. for each s_i' in S' ▷ check every subscriptions in new indexing set
11. { if ($count = k$)
12. then
13. { if ($j \neq m$)
14. then ▷ again grouping by next predicate
15. {
16. Procedure Grouping (p_{j+1}, S_temp, $count$)
17. }
18. return group index and create new group index
19. $S\_temp = \phi$
20. $count = 0$
21. }
22. $S\_temp = S\_temp \cup \{S_i'\}$ ▷ add S_i' into S_temp
23. $count = count + 1$
24. }
25. if ($j \neq m$)
26. then
27. {
28. Procedure Grouping (p_{j+1}, S_temp, $count$)
29. }
30. return the last group index

Figure 4: Pseudo code of SGIM grouping method

6.1 Experimental Methodology

One of the main challenges in evaluating pub/sub system is lack of real-world application data. However, previous work shows that in most applications events and subscriptions follow uniform or Zipf distributions. For comprehensiveness, we did our experiments with both of these distributions. We use subscription evaluation cost (in short, *evaluation cost*) like [10] as the metric of interest which is the average number of predicates that the system evaluates in matching an event to the subscriptions. We believe that it can well reflect the efficiency of the evaluation process.

Input:
1. An index G points to a set of indexes $\{G_1, \cdots G_{n-1}, G_n\}$ where n indicates the no. of indexes. Each G_i also points to a set of group indexes or a single group S'
2. A set of predicates P in S, $P = \{p_1, p_2 .... p_{m-1}, p_m\}$ where m indicates the number of predicates in a subscription. In our case, $m = 2$
3. p_j indicates we are searching by which predicate and initialy $j = 1$
4. An event E that contains a set of predicates $P' = \{p_1', p_2' .... p_{k-1}', p_k'\}$ where k indicates the no. of predicates in an event
5. A global result subscription set $result\_G$. Initially, $result\_G = \phi$

Body:
6. Procedure Searching (p_j, G, E, $result\_G$) ▷ search event E in G
7. for each G_i in G ▷ check each group for desired subscriptions
8.{
9. if (G_i may contain E) ▷ it depends on the p_j boundary of G_i
10. then
11. {
12. if ($j \neq m$) ▷ search in G_i by using p_{j+1}
13. then
14. {
15. Procedure Searching(p_{j+1}, G_i, E, $result\_G$) ▷ recursively call
16. }
17. else then ▷ already found the group
18. {
19. $S\_temp = S'$
20. $result\_G = result\_G \cup S\_temp$
21. for each p_i' in P'
22. {
23. for each s_r' in S_temp
24. {
25. if ($s_r'.p_i'$ *doesn't match* $E.p_i'$) ▷ subscription does not satisfy the event
26. then
27. {
28. $result\_G \setminus \{s_r'\}$ ▷ delete this subscription from result set
29. $S\_temp \setminus \{s_r'\}$ ▷ delete this subscription from temporary set
30. }
31. }
32. }
33. }
34. }
35.}

Figure 5: Pseudo code of SGIM event matching method

The parameters used in synthesizing the workload ((i.e., the set of subscriptions and events) are listed in Table 2. For each parameter setting, we repeat the experiments for multiple times and the results reported in the paper are based on the average over these runs. We used three predicate cases for comparison.

Table 2: Workload parameters in the experiments

Parameter	Description	Value
N_s	Total no. of Subscriptions	5K ~ 100K
$\min_p$	Minimum no. of predicates in a subscription	2
$\max_p$	Maximum no. of predicates in a subscription	3 ~ 10
N_e	No. of events	5000

6.2 Experimental Results

First we compared naïve, sequential, sub-order and forwarding algorithms to SGIM with three predicate cases using uniform distribution. The experimental results of evaluation cost for different algorithms are plotted in figure 6, 7 and 8.

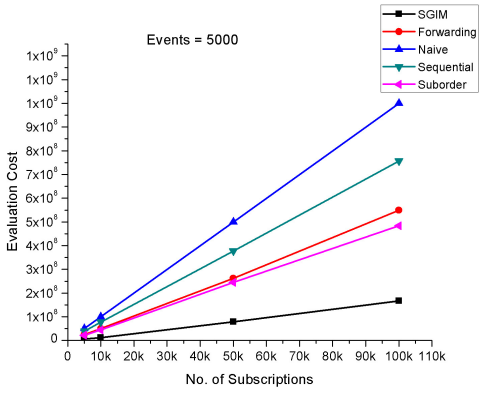

Figure 6: Evaluation cost vs. No. of subscriptions (Uniform distribution and predicate case 1: $mPV < data < MPV$ **)**

From figure 6, we can observe that for predicate case 1, *SGIM* performs better than all other algorithms. For example, with 50K subscriptions and 5000 events, the *naïve*, *sequential, forwarding*, *sub-order* and *SGIM* evaluate 5×10^8 , 3.7×10^8 , 2.6×10^8 , 2.4×10^8 and 7.8×10^7 predicates respectively. Thus *SGIM* reduces evaluation cost by 86%, 82%, 75%, and 76% as compared to *naïve, sequential, sub-order* and *forwarding* respectively.

From figure 7, we can observe that forwarding performs better than SGIM and others for predicate case 2. But, it requires more space for indexing all predicate values, while CGIM makes an index for groups. If we index all sub-predicates, equivalent performance will be achieved.

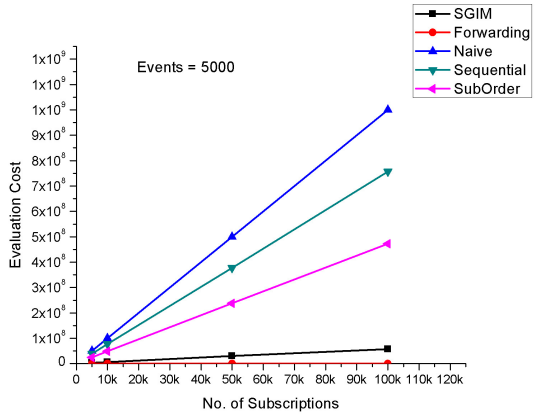

Figure 7: Evaluation cost vs. No. of subscriptions (Uniform distribution and predicate case 2: $mPV > data \,||\, MPV < data$ **)**

For predicate case 3 as shown in figure 8, SGIM performs better like predicate case 1 as compared to other algorithms. Notice that predicate case 3 can also represent events as range values. That is here events and subscriptions are ranged predicates. And also SGIM performs very well to handle range events.

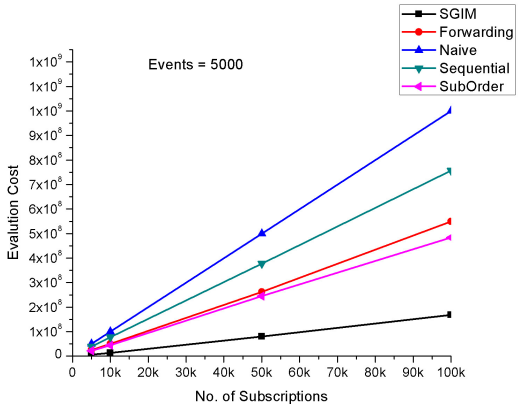

Figure 8: Evaluation cost vs. No. of subscriptions (Uniform distribution and predicate case 3: $mPV > data_1$ && $MPV < data_2$ **)**

Next we repeat the experiments with the same parameter settings, like figure 6 with predicate case 1 except that the distribution now follows Zipf (with exponent $\alpha = 0.8$) rather than the uniform distribution. The experiment results are plotted in Figure 9, which exhibits exactly the same trend as in Figure 6.

Above experiments clearly shows that our SGIM algorithm performs better (in case of uniform and Zipf distribution) than existing ones in terms of efficiency and scalability.

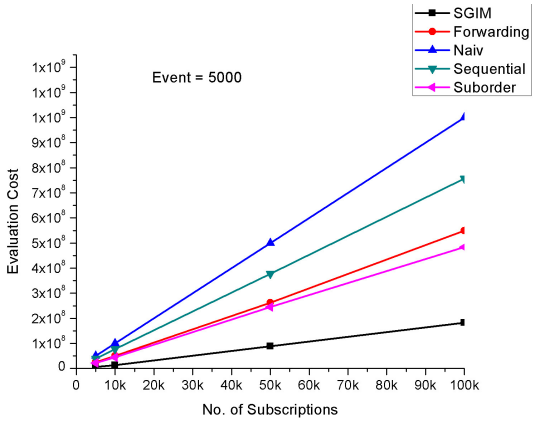

Figure 9: Evaluation cost vs. No. of subscriptions (Zipf distribution $\alpha = 0.8$ **)**

7. Conclusion

In this paper, we propose a framework of Sensor-Cloud connection to utilize the ever-expanding sensor data for various next generation community-centric sensing applications on the Cloud. It can be seen that the computational tools needed to launch this exploration is more appropriately built from the data center "cloud" computing model than the traditional HPC approaches or Grid approaches.. We propose a content-based pub-sub model to enable this framework. Also to deliver published sensor data or events to appropriate users of Cloud applications, we propose an efficient t and scalable event matching algorithm called *Statistical Group Index Matching (SGIM)* which targets range predicate case. We evaluated its performance and compared

with existing algorithms in a Cloud based ubiquitous health-care application scenario. Furthermore, our framework also addresses the issue of internetworking Cloud providers via a VO based dynamic collaboration for handling burst resource demand In future, we will study this issue more elaborately and try to find out an efficient resource negotiation algorithm to make the dynamic collaboration more efficient and faster.

8. ACKNOWLEDMENT

The research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2008-(C1090-0801-0002)).

9. REFERENCES

[1] Weiss. A. 2007. Computing in the Clouds. netWorker, 11(4):16-25, ACM Press.

[2] Buyya R. Yeo C. S. et al. 2008. Market Oriented Cloud Computing: Vision, Hype and Reality for Delivering IT Services as Computing Utilities. In Proc. of 10th IEEE Conference on HPCC'08, 25-27 September.

[3] Lim H. B., Teo Y.M., Mukherjee P., Lam V.T. et al. 2005. Sensor Grid: Integration of Wireless Sensor Networks and the Grid, In Proc. of the IEEE Conf. on Local Computer Networks

[4] Harris D. 2008. Managing Editor, On-Demand Enterprise. The Grid Cloud Connection: Compare and Contrast. http://www.on-demandenterprise.com/features/The_Grid-Cloud_Connection_Pt_I_Compare_and_Contrast_30634819.html

[5] Eugster P. Th., et al. 2003. The many faces of Publish/Subscribe. In ACM Computing Surveys Vol. 35(2): 114 – 131.

[6] Souto E., et al. 2005. Mires: A publish/subscribe middleware for sensor networks. In ACM Personal and Ubiquitous Computing, 10(1): 37-44.

[7] Krishnamurthy S. 2006. TinySIP: Providing Seamless Access to Sensorbased Services. In Proceedings of the 1st International Workshop on Advances in Sensor Networks (IWASN).

[8] Hall C. P., Carzaniga A., Rose J. and Wolf A. L. 2004. A content-based networking protocol for sensor networks. Department of Computer Science, University of Colorado, Technical Report.

[9] Hunkeler U., Truong H. L. and Stanford-Clark A. 2008. MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks. In IEEE Conference on COMSWARE

[10] Liu Z. et al. 2007. Scalable Event Matching for Overlapping Subscriptions in Pub/Sub Systems. In Proc. of DEBS'07, ACM Press, Toronto, Canada

[11] Carzaniga A., Wolf A. L. et.al. 2003. Forwarding in a Content-Based Network, In Proceedings of ACM SIGCOMM 2003. p. 163-174. Karlsruhe, Germany.

[12] Wu K., Chen S. and Yu P. S. 2004. VCR Indexing for Fast Event Matching for Highly-Overlapping Range Predicates. ACM Symposium on Applied Computing, Nicosia, Cyprus.

[13] Buberndorfer K. and Thomson W. 2006. Resource Management Using Untrusted Auctioneers in a Grid Economy. In Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)

[14] Pathan A. K., Broberg J., et al. 2007. An Architecture for Virtual Organization (VO)-Based Effective Peering of Content Delivery Networks. In Proc. of UPGRADE-CN'07. ACM Press, California, USA.