Performance evaluation of a web service enabled smart metering platform

Stamatis Karnouskos, Per Goncalves da Silva, and Dejan Ilic

SAP Research (www.sap.com)
{stamatis.karnouskos|per.goncalves.da.silva|dejan.ilic}@sap.com

Summary. A key issue for the success of a smart grid is the capability to accommodate efficient smart metering. Following the trend towards timely monitoring of energy consumption and production via Internet related technologies and in-network metering platforms, we need to investigate performance-related aspects of smart metering and how they affect the overall operation. We present here our experiences implementing a prototype framework for smart metering, discuss on some of its aspects, and evaluate its performance.

Key words: smart grid, web service, smart metering

1 Motivation

The emerging Internet of Energy [2], and more specifically its core entity, i.e. the Smart Grid, is a highly dynamic complex ecosystem of energy production and consumption parties that heavily use Information and Communication Technologies (ICT) in order to be more efficient compared to current traditional operation. Additionally, the Smart Grid enables the creation of new innovative services based on the bidirectional interaction of its stakeholders. The formation of new relationships between energy providers, distributors, dealers, and customers who themselves can act as producers (prosumers), has dramatically increased the complexity of the energy market.

Recent market statements for the smart-grid era, even considered with a grain of salt, provide some hints on the expected growth and business significance: Marie Hattar, vice president of marketing in Cisco's network systems solutions group, estimated in 2009 that the smart grid network will be "100 or 1000 times larger than the Internet". Similarly, Vishal Sikka, CTO of SAP, stated in 2009 that "The next billion SAP users will be smart meters".

Linking networked embedded systems (smart meters, energy control units, etc.), handling security and trust, modeling and transacting for highly distributed business processes, developing market-driven mechanisms for load balancing, proactive planning of system load profiles using derivatives, development of new business and market models, allowance for planning and scheduling, and assurance of interoperability are just a few of the topics that need to be specially defined and developed in this area [3].

The Internet Protocol (IP) is seen as one of the key technologies [1] that has big potential for the smart grid domain, including smart metering. It is expected that metering aggregation points, e.g. concentrators, will communicate over IP with online metering platforms and submit the collected metering measurements. Smart meters could also periodically connect and report their data not only to a single platform, for example, for billing, but also to multiple online services that could provide added-value [3]. Generally, due to the emergence of IP everywhere, such as the 6LoWPAN [4], it will be possible for any networked (embedded) device (meter, laptop, TV, etc.) to attach to the global IP network and report its energy consumption or production.

2 Smart Metering

The true power of SmartGrids can be realized once fine-grained monitoring, that is, metering of energy consumption or production, is in place. The promise of an Advanced Metering Infrastructure (AMI) is that we will be able to measure, collect, and analyze energy usage from advanced devices, such as electricity meters, gas meters, and/or water meters, through various communication media on demand or on a predefined schedule. Today, many utilities have already deployed or are currently deploying smart meters in order to enable the benefits of the AMI. A typical example is the world's largest smart meter deployment, which was undertaken by Enel in Italy and installed over 27 million smart meters to its entire customer base. AMI is empowering the next generation of electricity network, as for example the one depicted in the SmartGrid [2, 5] vision. Smart meters will be able to not only measure and report energy consumption in a timely manner, but also, in cooperation with online services or other devices, possibly provide management capabilities or information to the local network. These smart meters will be multi-utility, and their services will be interacting with various systems, not only for billing but for other value-added services as well [3].

We envision an infrastructure that will follow the Software as a service (SaaS) approach, where software vendors may host the application on (distributed) Internet servers and provide access to the value-added servers via a variety of media, such as on mobile devices, web portals, etc. While SaaS was initially widely deployed for sales force automation and Customer Relationship Management (CRM), its use has become commonplace in businesses for tasks such as computerized billing, invoicing, human-resource management, service-desk management, and sales-pipeline management, among others; we consider this approach to also be interesting for the SmartGrid era.

Within the scope of SmartHouse/SmartGrid (www.smarthouse-smartgrid.eu) and NOBEL (www.ict-nobel.eu) projects, we are defining and implementing a smart metering infrastructure that would glue heterogeneous systems and provide them common smart metering services. Although the concepts have been tested in a laboratory, the earliest real-world trials, which will start in mid-2010, are expected to deliver more results and hands-on experiences. An overview of

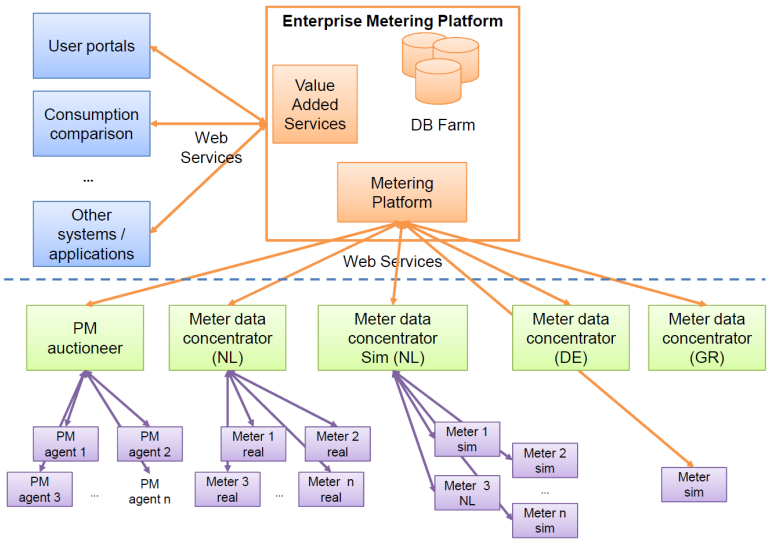

Fig. 1. Overview of smart metering in SmartHouse/SmartGrid project

the first trials is depicted in Figure 1. A commonality among them is the enterprise integration that is done towards two directions, (i) the metering data is reported for traditional business purposes, e.g. billing, and (ii) value-added services can be realized in conjunction with the context-specific info in each location.

The Internet-based metering platform features several components that implement the necessary services, such as MeterReading service to report real-time measurements, and is hosted in an Internet server. Currently, the main way to communicate among the platform and the different metering data-collections points is via web services [6], although in the future we envisage to experiment also with REST (Representational State Transfer) approaches. As such a concentrator, a smart meter or any other metering data entity can contact the necessary web service and submit the collected data.

3 Metering Platform Implementation

A metering service was realized as a web service. This service is used to submit the measurements acquired by the metering point to the platform. The smart meter web service is defined as a stateless Enterprise Java Bean (EJB). An EJB is a server-side component used to encapsulate business logic. The EJB is responsible for managing database operations for the insertion, update, deletion, and querying of meter reading data. The clients communicate with the server through the Simple Object Access Protocol (SOAP), a standard web service protocol used for exchanging the messages between clients and servers.

A key issue in the scalability of any smart metering platform is its ability to handle large numbers of requests, that is, the volume of smart meter readings should be handled in a timely manner by one or more instances of the platform (for load balancing). For simplicity, we assume that each platform instance may be hosted on a server, and by evaluating the limits of it, we can get a good indication of how many servers would be needed to reliably handle the targeted volume of data (scalability considerations). To this end, the time taken to handle a metering insertion request was measured at different stages of the incoming request life-cycle.

As can be seen in Figure 2, when a request arrived at the Application Server (realized by JBoss in our prototype), the appropriate EJB method was called and the data was inserted into the database. Subsequently, control propagates back to the server, which completed the request by sending a response to the client. The time required to execute these three stages, the total request time, EJB time, and database insert time, was measured.

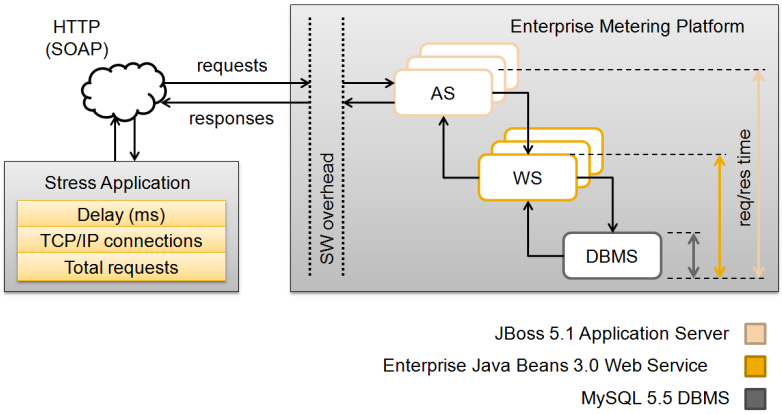

Fig. 2. Web service enabled Smart Metering

In order to realize the performance tests, we developed a prototype stress test application (in Java) that employed raw socket connections to communicate with the server and sent the generated meter readings. The data was wrapped in a SOAP message and sent via a POST request to the web service's URL address. The response from the server was acknowledged but no further processing was performed. The application enabled the tweaking of its functionality via three parameters:

- Number of sockets to use: a number of active connections established between the client and the server;
- Message delay: a wait time (in milliseconds) before the client sent the next request; and

– Total number of messages: the total number of the requests to be sent to the server. This value is independent to the number of the connected sockets.

From the operational point of view, the application generated requests and called the respective web service on the server at the specified interval until the total number of requests was reached. If the application was configured to use more than one socket connection, the requests were sent using each available socket in a Round-Robin fashion. Each request contained the exact message that a real metering point would submit (payload in XML), but in our prototype we generated these automatically and populated them with dynamic data, such as measurement value, meter ID, etc.

In the metering platform depicted in Figure 2, we can see that the web service layer (deployed with EJB 3.0) could have multiple parallel instances. Since the metering platform was able to handle parallel requests, the new Session Bean instances would be initialized if the metering platform started to receive data from multiple TCP connections. As with the socket connections, for every web service request a new thread would run on the server side, but an overall limit was defined within the Session Bean pool size. As we note later, this may be one of the bottlenecks towards achieving higher performance, which might be resolved with the usage of multiple application servers.

4 Performance Tests

Our initial aim was to investigate the ability of an online metering platform (hosted on a single server) to handle a relative heavy load of metering requests. This was done by stressing the server, that is, sending as many requests as possible, under different client configurations, in order to better understand the server behavior under peak load. This should provide some insight on how servers may be configured to reliably handle large numbers of requests, such as from one million smart meters. To achieve this, three scenarios were defined in terms of the number of workstations sending requests to the server and the number of sockets used by each workstation. The stress test application was used on each workstation to generate the load. Table 1 provides an overview of each scenario, its parameters, and performance. For all of the scenarios, we did not set any delays between the requests generated by the stress application, that is, generating (a lot) more requests than the server could handle.

Table 1. Performance Test Overview

Scenario ID	**clients**	**sockets/ workstation**	**requests/ workstation**	**requests/ second**	**requests/ 15 Min**
1	1	1	10000	435	391500
2	1	10	10000	672	604800
3	2	5	5000	769	692100

The tests were performed with several clients running Ubuntu Linux 9.10, Windows Vista, and Windows XP. On the server side, where the measurements were made, we used a COTS machine with an Intel Core Duo 6600 (2x2.4GHz) CPU, 4x2GB DDR2 667MHz memory, and a gigabit (one hop) Ethernet connection between the clients and the server. The server was running Ubuntu 9.10 64bit (2.6.31-21-generic kernel), with application server JBoss 5.1.0.GA and MySQL 5.1.37 DBMS. It should be noted that JBoss can be configured to deploy different server profiles, consisting of different service and module configurations. For the purposes of this experiment, the default server profile, which ships with the JBoss application server, was used. However, the Tomcat component of the JBoss server had to be configured with the maxKeepAliveRequests parameter on the HTTP/1.1 connector set to "-1" in order not to limit the the number of requests that can be made from a single connection.

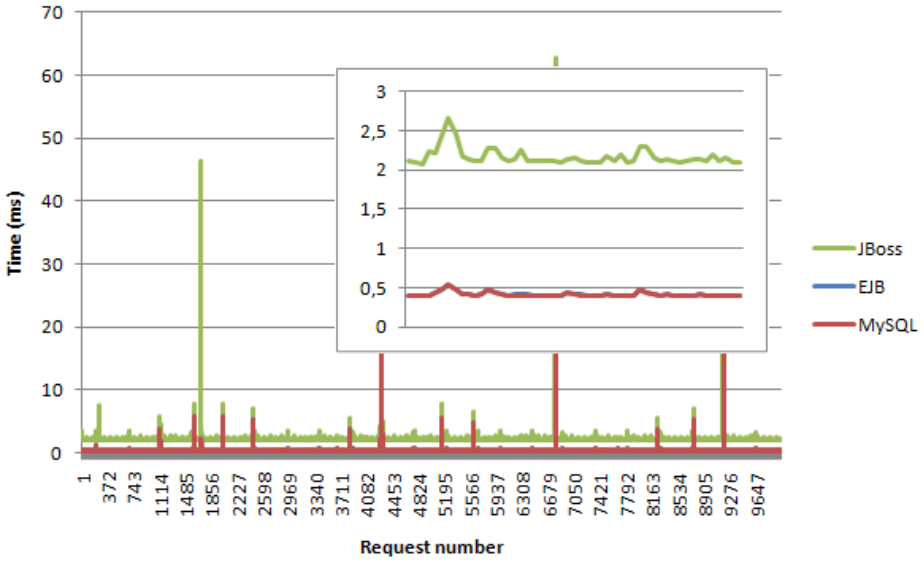

Fig. 3. Scenario 1 – Total request handling time

Scenario 1 is depicted in Figure 3, where we see the request handling times for each request. The EJB time is almost entirely comprised of the database insert time (both graphs overlap in Figure 3). We can see that there iss a big difference between total request handling time and the EJB time. This is probably due to overhead related to processing of the SOAP message, both at the request side (extracting the required information to perform the database insert) and on the response side (sending an empty SOAP response). Another interesting observation are the periodical spikes in message response time. As it can be seen in Figure 3, there are four peaks in response time, and a few smaller peaks

throughout. This may be the result of reoccurring operating system or database procedures.

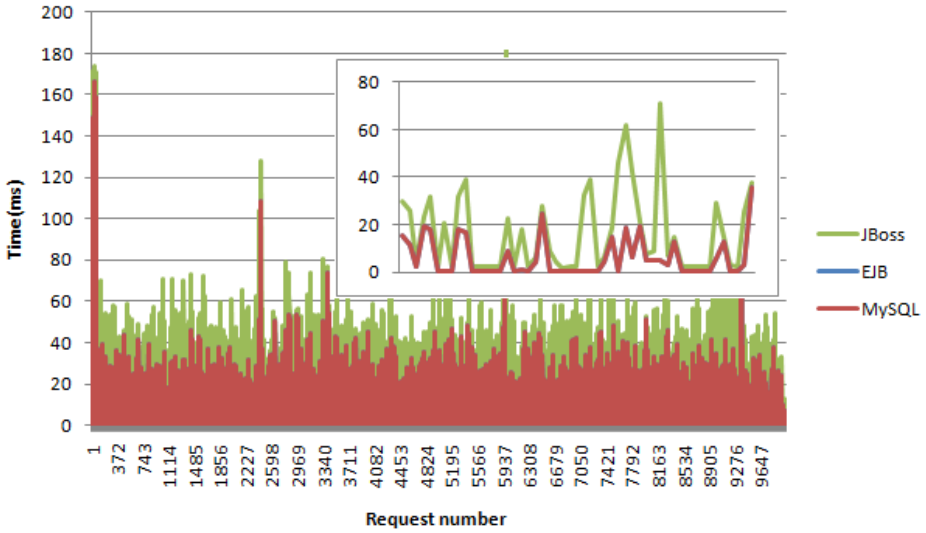

Fig. 4. Scenario 2 – Total request handling time

Scenario 2 is depicted in Figure 4, and demonstrates the response time of the server for requests made from one workstation using 10 sockets (in contrast to scenario 1 where we had only 1 socket). As it can be seen in Table 1, there is a clear gain in request throughput against scenario 1, while also sustaining a clear increase in response time (as shown Figure 4). The gain in throughput is a consequence of the use of multiple sockets, and thus multiple threads, resulting in the requests being handled in parallel. So, instead of one request being handled (scenario 1), ten requests are handled at the same time (scenario 2).

Figure 5 shows a comparison of the load duration curves for scenario 2 and scenario 3. As expected, the difference in insignificant, since the overall load is similar; the only difference is that in scenario 3 the composite load comes from two different workstations. However, as we can see, a higher throughput was achieved in scenario 3, most probably due to variable network conditions or JBoss AS internal management.

5 Discussion and future directions

We have seen that in the prototyped smart metering platform, a large portion of time is spent on internal processing happening at the Application Server itself. As shown in Figure 3, the total request/response time for one socket connection was approximately four times longer than the DB INSERT operation. This

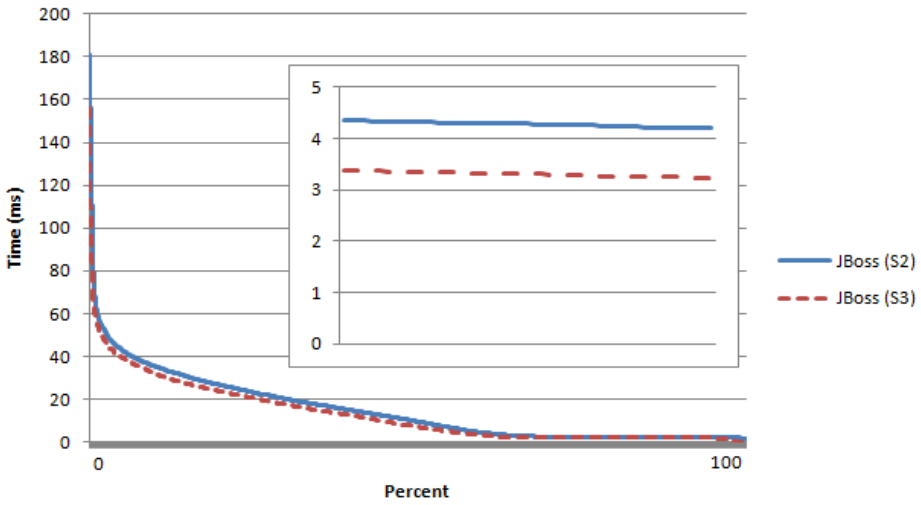

Fig. 5. Scenario 2 vs. Scenario 3 load duration curve

difference was possibly the first sign that the application server load should be balanced over multiple application servers. Therefore, further possible performance enhancements should be explored in order to reduce that time, for example via the usage of multiple application platforms running as front-end. However, more aggressive performance-related strategies might provide better results, such as usage of in-memory DBs or strategic (on-demand or periodic) committing to the DB.

As already noted and also depicted in Table 1, the indicative performance depends on many factors. With the current configuration, our scenarios would accommodate in a per-15-minute measurement window approximately 692k (scenario 3) measurements. This gave some early indication on the infrastructure that needed to be in place to accommodate millions of meters in this time interval. However, we have to point out here that these end-metering points could also be concentrators that then would further increase the total number of last-mile end-points (smart meters results are grouped at concentrator level) whose data could be collectively reported.

Our test generates (on client side) and pushes (to server side) a high amount of generated data; as such, the communication throughput is limited by the TCP receive window (server side). If we take a closer look into the transport layer (Figure 6), it can be seen that the server side reaches its input buffer limits (TCP window size). Thus, TCP window scaling (RFC 1323), that is, to increase the TCP receive window size above its current value (Windows default is 65535 bytes), is an option. The increase of the TCP window on the server side helps to not exceed the capacity of the receiver to retrieve data (flow control). As we can see in Figure 6, TCP window updates were very common, and also TCP ZeroWindow occurred during our tests. This was due to the fact that our client generates (many) more requests than those the server can handle. Thus, the

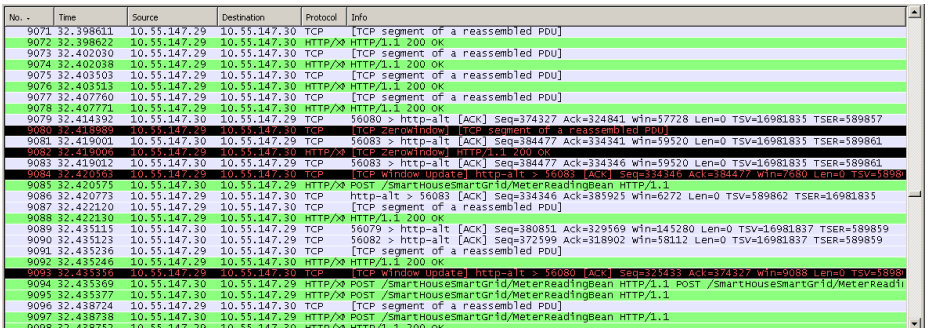

Fig. 6. Communication analysis: TCP Window scaling

client will continue generating data, but it will be kept in output buffer until the server updates its window size to equal (or greater) value of the message size to be sent. These steps will be repeated over and over again until all data is transmitted.

Apart from the initial performance evaluation presented here, several directions can still be assessed both in the laboratory environment and in real world conditions. With respect to metering data exchange, the time penalties in submitting single vs. multiple metering values from one or multiple locations needs to be further evaluated. The reliability needs to be investigated, especially over unreliable or congested channels. The metering payload and its correlation to processing and transmission time need to be further evaluated. Furthermore, we mostly assume best-effort network, therefore we wanted to take a closer look at network performance aspects, such as possible optimizations on the communication strategy: create connections per message, per client, strategy to manage connection time, such as one connection open for multiple measurement submissions, etc. The usage of approaches that provide some guarantees, such as WS-ReliableMessaging, or security, such as exchange of signed/encrypted measurements, would also be of interest.

For the metering platform, we want to further investigate issues related to throughput, that is, performance measurements related to number of measurement messages processed, resource consumption e.g. CPU, memory, etc, message processing time (from acceptance to storage), as well as storage performance (for storing and retrieving). Scalability of the platform (clustering, etc.) and/or its components is also an issue, especially considering the heavy load that near real-time metering might pose. End-to-end service performance (for example, from metering of data up to end-user display) would also enable us to see if and how real-time services can be provided.

6 Conclusions

We have presented our experiences in prototyping an online smart metering platform that can communicate via web services with the metering points and

collect the measurements. Initial evaluation shows that the concept and technology approach are sound and that we can achieve a high number of measurements, even with COTS hardware and software, and with no significant performance tweaking. Although there are many aspects to be evaluated, here we focus on a service-enabled infrastructure and evaluate the performance of a simple prototype platform for acquiring and storing high numbers of metering measurements.

Nowadays, many commonly refer to high-resolution metering, which is considered in a "15-minute" period. However, in the near future we will move not only towards real-time metering but also expand on the notion of "meters", since any of the billion Internet of Things envisioned devices could be acting as a "meter". This trend will pose some significant requirements to metering platforms in order to be able to accommodate all measurements in a timely manner. We have shown that simple prototype solutions as ours can achieve considerable performance. However, requirements for more reliability and scalability will increase in the future.

Acknowledgment

The authors would like to thank the European Commission for support and the partners of the SmartHouse/SmartGrid (www.smarthouse-smartgrid.eu) and NOBEL (www.ict-nobel.eu) for fruitful discussions.

References

[1] NIST framework and roadmap for smart grid interoperability standards. National Institute of Standards and Technology (NIST) NIST Special Publication 1108, National Institute of Standards and Technology, Jan. 2010.
[2] Federation of German Industries (BDI). Internet of Energy: ICT for energy markets of the future. BDI publication No. 439, Feb. 2010.
[3] S. Karnouskos and O. Terzidis. Towards an information infrastructure for the future internet of energy. In *Kommunikation in Verteilten Systemen (KiVS 2007) Conference*. VDE Verlag, 26 Feb. - 02 Mar. 2007.
[4] Z. Shelby and C. Bormann. *6LoWPAN: The Wireless Embedded Internet*. Number ISBN: 978-0-470-74799-5. Wiley, Nov. 2009.
[5] SmartGrids European Technology Platform. Smartgrids: Strategic deployment document for europe's electricity networks of the future, Apr. 2010.
[6] C. Warmer, K. Kok, S. Karnouskos, A. Weidlich, D. Nestle, P. Selzam, J. Ringelstein, A. Dimeas, and S. Drenkard. Web services for integration of smart houses in the smart grid. In *Grid-Interop - The road to an interoperable grid, Denver, Colorado, USA*, Nov. 17-19 2009.