Benchmarking *Internet of Things* Solutions

Ashok Joshi[1(✉)], Raghunath Nambiar[2], and Michael Brey[1]

[1] Oracle, Redwood shores, USA
{ashok.joshi,michael.brey}@oracle.com
[2] Cisco, San Francisco, USA
rnambiar@cisco.com

Abstract. The Internet of Things (IoT) is the network of physical objects accessed through the Internet, as defined by technology analysts and visionaries. These objects contain embedded technology allowing them to interact with the external environment. In other words, when objects can sense and communicate, it changes how and where decisions are made, and who makes them. In the coming years, the Internet of Things is expected to be much larger than the internet and world-wide web that we know today.

1 Introduction

The *Internet of Things* (IoT) promises to revolutionize various aspects of our lives. Until recently, most of the internet activity has involved human interaction in one way or another, such as web browser-based interaction, e-commerce, smart-phone apps and so on. Recently, there has been a lot of interest in connecting devices of various sorts to the internet in order to create the "internet of things" for delivering dramatic benefits relative to existing solutions. For example, network-enabled home health-care monitoring devices hold the promise of improving the quality of healthcare while reducing costs. Connected energy monitoring systems enable more efficient utilization of energy resources and reduce global warming. Traffic management systems aim to reduce congestion and decrease automobile accidents. It is clear that such systems can deliver tremendous value and benefits over traditional solutions.

IoT has fueled massive investment and growth across the entire spectrum of industry and public sectors. Companies such as Oracle [1], Cisco [2], Intel [3], Microsoft [4] and others, are investing heavily in IoT initiatives in order to address the computing and infrastructure requirements of this market.

Though the IoT industry is still in its infancy, based on the number and variety of solutions being proposed, it seems clear that IoT solutions will play a very significant role in the coming years. Although there has been a lot of IoT marketing and promotional activity from small and large vendors alike, there is a paucity of proposals to measure the performance of such systems. At best, some of the proposals [5, 6, 7] either measure the performance of a specific component and extrapolate the results to a broader

This paper proposes an approach to benchmarking the server-side components of a complete Internet of Things solution.

T. Rabl et al. (Eds.): WBDB 2014, LNCS 8991, pp. 29–36, 2015.
DOI: 10.1007/978-3-319-20233-4_4

IoT solution [5], or propose a very broad and comprehensive benchmark *framework* that encompasses performance, public policy and socio-economic considerations [7].

In contrast, our approach is much more specific and focuses exclusively on the performance metrics of an IoT system (in particular, on the server-side components of an IoT solution) using a representative workload. We believe that a systematic approach to performance benchmarking of IoT systems will help users evaluate alternative approaches, and ultimately accelerate adoption; in other words, our approach is workload and solution-centric. A well-crafted performance benchmark also serves as a catalyst for vendors to improve the performance as well as the overall usability of the solution, which should benefit the IoT consumer as well.

In the rest of the paper, we describe the components of a typical IoT solution, then describe a representative use-case, followed by a straw-man description of a benchmark intended to measure the performance (throughput, latency, etc.) of such a system. Admittedly, this is a straw-man proposal; we explore how this benchmark might be applicable to other use-cases (with modifications) and also describe some of the limitations of our proposal and areas for future work. Significant additional work is necessary in order to create one or more formal benchmarks for IoT systems.

2 IoT Solution Architecture Components

Although the scale of IoT solutions is enormous in comparison to current internet usage, in this paper, we address only the essential architecture of an IoT solution in the context of benchmarking the server components. The following discussion presents a simplified view of the system as a small number of distinct, essential components. These architecture components and their relationships are illustrated in Fig. 1 and described below in more detail.

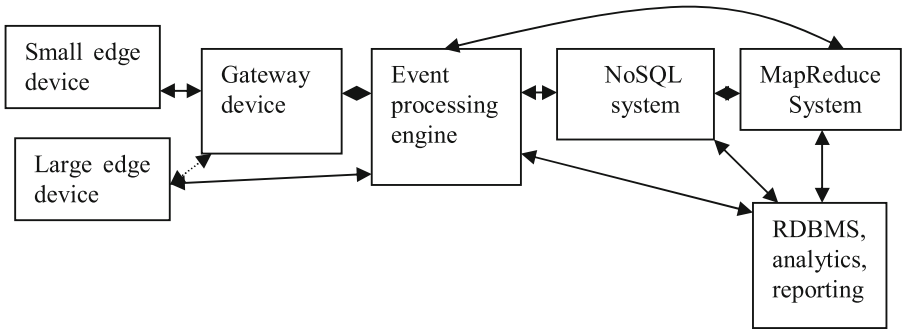

Fig. 1 Essential components of a typical IoT architecture.

At the "device" edge of the network, we have *edge devices* and software that controls the devices. Quite often, the devices are relatively inexpensive and have limited computing and network capabilities (*small edge devices*) – temperature sensors in office buildings, various sensors in a vehicle, blood pressure and blood glucose

monitors are all examples of such devices. Mobile phones, computers in vehicles are also examples of edge devices which have better computing and local storage capabilities than small edge devices. Small edge devices are primarily intended to capture and send information from the object being monitored, to the central repository; secondarily, they may also receive data to make appropriate changes (e.g. a building temperature sensor might receive a directive to shutdown the air conditioning system in the event of a fire). More capable edge devices like mobile phones (large edge devices) typically provide a richer set of capabilities beyond simple monitoring, and can have bi-directional exchange of information with the central data repository.

Depending on the number and capability of edge devices in the deployed topology, edge devices (such as sensors) connect to a *gateway device*, organized as a multi-level hierarchy, with a set of small devices being served by a gateway device. The gateway device with sufficient computing, storage and network connectivity is essentially a "concentrator" that can process data and exchange information between the small devices and the data center. A gateway device communicates with a relatively small set of edge devices (typically less than one hundred) on one side and the data center on the other. For example, a set-top box (gateway device) might serve as the connectivity "hub" for all the sensors and devices in a home. Or a laptop/desktop-class device might serve as the connectivity hub for sensors on a set of machines and robots on a factory floor. A gateway device is connected to the edge devices on one side, and to the data center (or other gateway devices in a multi-stage hierarchy) on the other side and enables efficient communication and data exchange between edge devices and the rest of the system. Note that in a large system, there might be multiple gateway stages, in order to simplify data management and communication in the system.

A large edge device may either be connected directly to the central repository, or indirectly through an intermediate gateway device, depending on the specific application scenario. For example, a mobile phone might connect directly with the data center, without any intermediate gateway device.

The other major component in the hierarchy is obviously the network. The network is not only responsible for communication, but it also facilitates or enforces attributes such as security, privacy, reliability, availability and high performance.

Mobile edge devices pose an interesting challenge since they may occasionally be out of range of a network connection. For example, an edge device mounted on a delivery truck may occasionally lose network connectivity when the truck enters a tunnel or is in a remote area. In order to avoid data loss while the device is disconnected from the network, it is necessary to temporarily store new data in the (large) edge device and send that information to the data center when connectivity is restored. Even in cases where temporary loss of network connectivity is not an issue, it may still be valuable to store some data locally on the edge device or gateway device and only synchronize it periodically with the data center, in order to minimize costs associated with network usage, since these costs can add up quickly, especially at IoT scale. Oracle Database Mobile Server [8] is an example of a technology that is well suited for data synchronization between edge device repositories and the data center. Lightweight, embeddable databases such as SQLite [9], Oracle Berkeley DB [10] and Java DB [11] are well suited for reliable, device-local data management.

Finally, the *data center* (centralized or distributed) receives data from the edge devices and processes it in various ways, depending on the application. Broadly speaking, data center processing can be classified into *event processing*, *high throughput data capture*, *device management*, *data aggregation, data analytics* and *reporting*.

We will use the term *event processing* to describe the processing that is required to respond to an event as quickly as possible. For example, if a sensor in a building sends information about a potential fire, then it is necessary to respond to that message as quickly as possible. Event processing typically involves various lookups (e.g. look up the location of the device, determine location of the closest emergency responders) as well as generating appropriate responses (e.g. send a message back to the sensor to switch on the sprinkler system, send an alert to first responders). Note that of all the messages being received from the edge devices, only a small fraction need an immediate response as described above. However, all the messages must be examined by the event processing engine in order to determine whether an immediate response is required.

Messages from edge and gateway devices need to be captured persistently in order to enable further analytics. Though message sizes are generally small, typically varying from few tens of bytes, to a few kilobytes, the sheer volume of devices involved and the rate of data capture usually implies that a scalable, persistent store to capture the data is required. Depending on the nature of data and the application, a NoSQL database system or a distributed file system such as HDFS might be used. For example, data related to medical information (e.g. blood pressure, heart rate and blood glucose readings) for patient monitoring might need to be stored in a NoSQL database for reliability and regulatory reasons. On the other hand, a distributed file system such as HDFS might be sufficient to store messages containing information from office building room temperature sensors.

A scalable database system (e.g. a NoSQL system) is also typically required to manage information about the edge and gateway devices. Such information typically includes the ID and owner of the device, location, model number and other details, software patch version etc. Occasionally, it is necessary to send updates or new information to the devices, for example, to deliver software upgrades, or change configuration settings. In an IoT environment, it is very likely that such information needs to be managed for hundreds of thousands to millions of edge devices; a NoSQL repository is ideally suited for this purpose.

Lastly, the data received from the devices needs to be aggregated and processed in various ways to identify trends, opportunities for optimization as well as to generate reports required for running the business. For example, data collected from vehicle brake sensors might be correlated with GPS locations and time-of-day in order to optimize delivery routes. Data from heart rate sensors and physical activity sensors might be used in order to recommend an appropriate exercise regimen for heart disease patients.

3 Benchmarking IoT

IoT is rapidly emerging as the “next generation” technology, and promises to revolutionize and improve various aspects of everyday life. There are literally hundreds of commercial and open source organizations in this broad and somewhat amorphous space.

Regardless of the kinds of solutions available, we believe that a systematic approach to measurement and benchmarking such systems will provide significant benefits to the user community and solution providers.

Attempting to define a comprehensive benchmark that deals with the various components in the end-to-end solution is a daunting task; in the following paragraphs, we describe a skeleton benchmark intended to address performance issues of some of the server-side components. The suggestions and recommendations outlined below have been derived from the authors' real-world understanding and experience with early deployments of IoT solutions. In the future, this work can be extended to define additional benchmarks to address other aspects of an IoT solution.

The proposed "strawman" benchmark is intended to model and measure some of the server-side components of a real-world scenario viz. to capture data from home health-care monitoring systems for millions of patients. This involves capturing health-related readings reliably and securely from large numbers of devices, responding rapidly to abnormal events, and analyzing vast amounts of data in order to benefit the patient community.

A *reading* from an edge device is typically short (tens of bytes) and contains information like device ID, timestamp, geo-location (if appropriate) and measurements of the variables that the sensor captures. For the purposes of a benchmark, a reading is 96 bytes of data, with a 16 byte key (device ID) and five 16-byte fields of data. Let us assume that each edge device (e.g. a heart rate monitor) captures a reading every 10 min. For a population of one million patients, this translates to a reading capture rate of 1667 readings/sec and 160 KB/sec. As soon as the reading is available, the edge device sends the data to the data center either directly or via intermediate gateway devices.

An event processing system processes every incoming reading. A small percentage (e.g. 1 percent) of the incoming readings will need a response immediately and every reading needs to be saved in a secure, persistent repository. The event processing system interprets the incoming reading and decides whether it requires an immediate response. This decision is based on the reading itself as well as information about the specific patient (this requires lookups in the patient database). For the purposes of the benchmark, let us assume that the decision involves looking up a single database that maintains per-patient information and a history of the recent three months' worth of heart-rate readings (about 13000 heart-rate readings, or 1.2 MB as well as the patient's recent medical history of 100 KB).

Regardless of whether they require an immediate response or not, all readings are stored in the patient history repository. Based on the previous description, for a patient population of one million patients, this translates to a repository that can manage approximately 1.3 TB of patient data.

Periodically, a batch job reads the heart rate readings history for each patient and consolidates multiple "similar" adjacent readings into a single reading. Since each sensor measures the heart-rate of the patient every 10 min, it is possible that a series of readings might have heart-rate readings that are within the normal range for the patient. For example, while the patient is asleep, the heart-rate might stay relatively constant for a significant interval of time. The batch job identifies such intervals and consolidates multiple readings at taken at 10 min intervals into a single "average" reading for a

longer interval. For example, if the patient's heart-rate doesn't fluctuate by more than 3 heartbeats per minute between two subsequent readings, these two readings will be "consolidated" into a single reading which has the average value of the two original readings and the time interval will be changed to be a 20 min span. This consolidation operation has the dual benefit of reducing the amount of storage required and also simplifying the job of the physician by consolidating similar, adjacent readings into a single reading over a larger interval. For the purposes of the benchmark definition, we assume that 50 % of the readings are consolidated into a single reading. This translates to a repository of 1.3 TB of patient data for a patient population of 1 million patients.

Operation Profiles. The message format and size, as well as the patient record size and content have been described earlier. The various operation profiles are as follows:

- Incoming reading received by the event processing engine. For each reading, the event processing engine looks up the related patient record. For 1 % of the messages (selected randomly), it generates an alert. Every message is appended to the related patient record. From a benchmarking perspective, this operation is intended to model the data capture throughput and data retrieval throughput of the system.
- Consolidating similar, adjacent heart-rate readings. A background job reads each patient reading, consolidates similar, adjacent heart-rate readings into a single reading and updates the patient record appropriately. The choice of whether to retain the readings that contributed to the consolidated reading or delete such readings is left up to the implementers of the benchmark. The consolidation operation should be able to consolidate 50 % of the readings within a 24 h window. The batch job must retrieve every record in the database within the same period. From a benchmarking perspective, this operation is intended to model the analytics and batch-processing aspects of the system.

A key premise of most IoT systems is the ability to respond in a timely manner to abnormal events. Secondly, the volume of readings and measurements from sensors can be enormous. The ability to filter the "signal" from the "noise" is critical to efficient operation of the system. These operation profiles attempt to capture these aspects in a benchmark.

Performance and Price-Performance Metrics. The benchmark measures the throughput of data capture (readings/sec) in the patient repository under the constraints described earlier. Further, data capture, response generation and data consolidation jobs must be run concurrently, since the system is intended to run 24X7. It is difficult to propose a price-performance metric for the benchmark, since several necessary components of the complete system are intentionally ignored in this proposal.

Other Considerations. This paper provides some recommendations for benchmarking the event processing and data repository components of a IoT system intended to capture and process data for a patient data monitoring system. It also suggests scaling rules for such a benchmark.

Several variants of the above approach are possible, depending on the specific IoT application being modeled. It may be necessary to vary some of the parameters as follows:

- The number of edge devices, typical size of a message from each device and the frequency of sending a message from each edge device.
- Event processing requirements.
- Number of historical messages to be retained (in this proposal, we recommended storing three months worth of readings).
- Size of the device record (e.g. the size of each patient record, in this proposal).
- Other parameters as relevant to a specific usage scenario.

From a benchmarking perspective, this proposal addresses the following essential components:

- Data capture from multiple sources (edge or gateway devices) - this measures "raw" data capture throughput.
- "Real-time" analysis of the input data for fast response to a subset of the incoming data – this measures the performance of the event processing engine as well as the throughput and latency of accessing reference data from the data store.
- Consolidation of the incoming data in order to conserve space – this measures batch processing (as a proxy for analytics processing) performance of the system.

We believe that server-side data capture, real-time response, and batch processing performance are common to a wide variety of other IoT usage scenarios such as climate control, vehicle monitoring, traffic management and so on. With appropriate changes to model a particular scenario, this proposal can be adapted to measure and benchmark the performance of the server components of such scenarios.

4 Conclusion

Internet-of-Things (IoT) represents a broad class of emerging applications that promises to affect everyday life in various ways. The scale of IoT applications dwarfs the use of the internet, as we know it today. Many commercial and open source players are investing heavily in solutions intended to address specific aspects of IoT; some of the larger players claim to provide a comprehensive, end-to-end solution.

For all these scenarios, we believe that a set of benchmarks, modeled on real-world scenarios, will provide a valuable yardstick to compare the relative merits and costs of such solutions and will have a significant beneficial impact on the IoT industry. In this paper, we have attempted to provide some guidelines on how one might go about designing such benchmarks. Significant additional work is needed in order to define formal benchmarks; hopefully, this proposal will inspire some of that future work.

References

1. http://www.oracle.com/iot
2. http://www.cisco.com/web/solutions/trends/iot/overview.html
3. http://www.intel.com/iot
4. http://www.microsoft.com/windowsembedded/en-us/internet-of-things.aspx

5. http://www.sys-con.com/node/3178162
6. http://www.probe-it.eu/?page_id=1036
7. http://www.probe-it.eu/wp-content/uploads/2012/10/Probe-it-benchmarking-framework.pdf
8. http://www.oracle.com/technetwork/database/database-technologies/database-mobile-server/overview/index.html
9. http://www.sqlite.org
10. http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html
11. http://www.oracle.com/technetwork/java/javadb/overview/index.html