Viability of Azure IoT Hub for Processing High Velocity Large Scale IoT Data

Wajdi H Halabi
whalabi@clemson.edu
Clemson University
Clemson, South Carolina, USA

Daniel N Smith
dnsmith@clemson.edu
Clemson University
Clemson, South Carolina, USA

John C Hill
jhill326@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Jason W Anderson
Jason.Anderson@partner.bmwgroup.com
BMW IT Research Center
Greenville, South Carolina, USA

Ken E Kennedy
ken.kennedy@bmwgroup.com
BMW IT Research Center
Greenville, South Carolina, USA

Brandon M Posey
brandon.posey@bmwgroup.com
BMW IT Research Center
Greenville, South Carolina, USA

Linh B Ngo
lngo@clemson.edu
Clemson University
Clemson, South Carolina, USA

Amy W Apon
aapon@clemson.edu
Clemson University
Clemson, South Carolina, USA

ABSTRACT

We utilize the Clemson supercomputer to generate a massive workload for testing the performance of Microsoft Azure IoT Hub. The workload emulates sensor data from a large manufacturing facility. We study the effects of message frequency, distribution, and size on round-trip latency for different IoT Hub configurations. Significant variation in latency occurs when the system exceeds IoT Hub specifications. The results are predictable and well-behaved for a well-engineered system and can meet soft real-time deadlines.

KEYWORDS

Massive workload generation, Emulated high velocity IoT data, Azure IoT Hub, Cloud performance

ACM Reference Format:
Wajdi H Halabi, Daniel N Smith, John C Hill, Jason W Anderson, Ken E Kennedy, Brandon M Posey, Linh B Ngo, and Amy W Apon. 2021. Viability of Azure IoT Hub for Processing High Velocity Large Scale IoT Data. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3447545.3451187

1 INTRODUCTION

Commercial clouds have been shown to be useful for massive computing and data-intensive workloads [6, 7]. In this paper, we study Microsoft Azure IoT Hub using a massive synthetic workload generated by the Clemson supercomputer. The workload emulates proprietary, large-scale data from many thousands of sensors in a large manufacturing facility[1] [3]. The use of sensors within manufacturing has rapidly increased due to initiatives such as Industry 4.0 that involve the collection and use of data throughout all aspects of the manufacturing process. Data can be analyzed historically, say, for predictive maintenance and for optimization of existing processes. Processing of data may require meeting a soft real-time deadline for managing problems on the manufacturing line.

We aim to characterize the workload that can be processed in the cloud in a fixed time period. Since IoT Hub is a shared resource, we want to know how performance may be impacted by workload [4]. We also aim to provide advice regarding the number and characteristics of cloud-connected sensors that can be connected to IoT Hub for soft real-time analysis.

2 BACKGROUND – AZURE IOT HUB

In this section we describe the Azure IoT Hub protocols, pricing, throttling limits, and partitions.

Protocols. Azure IoT Hub uses HTTPS, AMPQ, or MQTT to facilitate communication between a sensor and the cloud. MQTT is the preferred protocol when using IoT Hub. It is a secure and lightweight protocol designed for Internet of Things connectivity and utilizes the publish-subscribe message model. Azure IoT Hub implements MQTT v3.1.1.

Each IoT sensor has a unique connection string between it and the IoT Hub. The connection string is the concatenation of the IoT Hub hostname, the unique sensor ID, and the shared access key. These connection strings map the cloud-side systems processing the messages to exactly one actively connected sensor, enabling secure messages. IoT Hub can route messages to other Azure services. IoT Hub provides *units*, which refer to the number of instances deployed on the cloud. Sensors are still registered to an IoT Hub and are not bound to any particular unit.

Pricing. Iot Hub is split into three *editions*, which vary in the limits available to users. It is further divided into Basic and Standard *tiers*, which share throttling limits, but Standard has more features. IoT

ICPE '21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451187

Table 1: Iot Hub Editions and Throttling Limits

Edition	Max. S2C[a] Send Operations	Max. Messages per Day
B1/S1	100 per sec[b]	400,000 per unit
B2/S2	120 per sec per unit	6,000,000 per unit
B3/S3	6000 per sec per unit	300,000,000 per unit

[a]Sensor-to-Cloud [b]This is 12 per sec per unit if it results in a higher value.

Hub is charged not based on consumption but at a flat rate based only on tier, edition, and days of provisioning.

Throttling Limits. Different editions have different throttling limits. For example, there are limits for daily message allowance, maximum aggregate messages per second, and maximum new connections per second. This information is summarized in Table 1.

Partitions. When creating a new IoT Hub using the online Azure portal, one can request between 4 and 32 partitions, with a default of 4. However, it is possible to request up to 128 partitions if one creates their IoT Hub using the Azure Command Line Interface. Note that partition count does not affect price. New messages are tagged to the end of a partition and stored for a specific retention time. Each sensor is deterministically assigned to a partition using its unique ID and, thus, partitions grow independently. Messages sent from a sensor ID go to the assigned partition and retain their order.

3 SOFTWARE AND CONNECTIVITY ARCHITECTURE

Here we describe the supercomputer environment, synthetic workload client, experiment loop and connection.

Palmetto Supercomputer. We utilized up to ten nodes from the Clemson Palmetto supercomputer to emulate thousands of sensors by executing clients across multiple cores and nodes. The Clemson network mimics the network of a manufacturing facility with a wide variety of applications, multi-level network policies and protocols, and different SLAs [5]. Palmetto has fast, direct access to the Internet, and there is typically minimal contention for outbound Internet traffic. Packets from Palmetto to the Azure East US 2 region follow a path connected by a shared 100Gb path to the Internet gateway, to a 10Gb link for commodity traffic, to the Microsoft internal network. 100 pings from Palmetto to a VM hosted in that region resulted in an average latency of 20ms and a standard deviation of 0.846ms.

Client Data Generator. The synthetic data generator is written in C++ to ensure a small memory footprint. The generator represents a single physical sensor that reads data periodically and sends it to IoT Hub. For example, ten instances of the generator running simultaneously simulate ten sensors. Parameters that differentiate data generation behavior are shown in Table 2. Each instance writes its own individual log file that contains message ID, send time, asynchronous callback receive time, and the callback status.

Experiment Loop. The client generates a message modeled after the specified parameters, immediately sends it to IoT Hub, and logs the send time. These messages are JSON strings with a size specified in bytes. The SDK creates a thread for every message callback, which asynchronously listens for a response. The number of individual messages is based on the runtime specified. Between each message, the main thread sleeps while the asynchronous thread waits to receive the response from IoT Hub and logs the receive time and message status. We measure round-trip latency, calculated using the logged send and receive times, with sub-millisecond precision. The status of a message can be *OK* (IoT Hub successfully processed the message), *destroyed* (IoT Hub could not or would not process the message and rejected it), or *other*. Time costs of other services that could be used to process the data from IoT Hub are not included. Some of the latency we observe is due to the SDK thread creation for the asynchronous message receipt. This is consistent with the measurement of the end-to-end latency in a production application.

Table 2: Synthetic Data Generator Parameters

Parameter	Range
Statistical Distribution	Constant OR Pareto
Inter-message Gap Time	10 ms - 1000 ms
Message Size	512B - 32,768B
Experimental Run Time	5 min - 90 min
Network Protocol	MQTT

Each instance of the client data generator is a sensor from Azure's perspective. As such, we refer to our data generator as a sensor for the remainder of this paper. The initial connection between the sensor and Azure requires setup and has higher than normal latency, so we disregard the first 5% of messages. Steady state latency measurements are representative of a production environment.

Experiments use the default of four partitions except when comparing the latency performance of different partition counts. We choose the shortest message retention time of one day.

Validating Data Generator. We validated the generator by confirming the intermessage gap times (IMT) followed the expected distribution by measuring the gaps between network packet send times. We specified either a constant distribution with a fixed IMT or a Pareto distribution with a variable IMT. A Pareto distribution models sensors with bursty send characteristics, which captures many real-world scenarios. Inputs into the data generator were the sending frequency, distribution parameters, and the message payload. We ran this validation on the Clemson Research Datacenter.

While the generator was sending messages, we tracked the sent packets using the sniffing tool `tcpdump` to accurately record the host timestamp of packets sent. For constant IMT, we calculated the median and standard deviation of the resulting message distribution. Here, we observed a standard deviation less than 0.5 ms. For the long-tailed Pareto distribution, we tested the observed timestamps with a Kolmogorov-Smirnov (KS) test to evaluate the goodness of fit of the resulting distribution. The KS test results showed that the generated IMTs were a good fit for Pareto distribution. The d-value was 0.03 and the p-values were .88.

4 EXPERIMENTAL STUDY

The experiments isolate and measure the effects on IoT Hub performance of various counts of client sensors, message sizes, intermessage gap times (IMT), and IoT Hub partition counts. By testing a large range of parameters we have a high confidence that a specific real world workload is represented in the experiments.

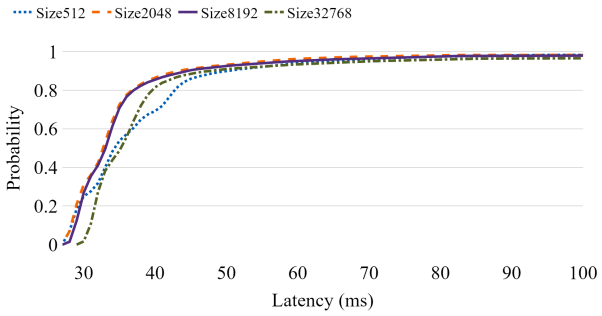

Figure 1: Latency CDF of 10 sensors, Edition 3, constant IMT of 200 ms.

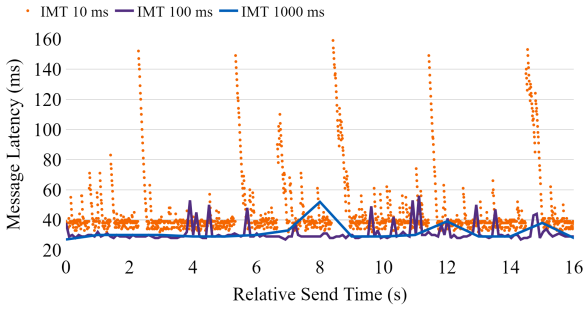

Figure 2: Latency of 1 sensor, 4 partitions, constant IMT distributions.

Effects of Different Message Sizes. We examined whether increasing the message size correlates with increased IoT Hub round-trip latency. To observe the effect of message size on latency, we conducted experiments using 512B, 2048B, 8,192B, and 32,768B messages sizes across the three editions of IoT Hub Basic tier. We test only the Basic tier since both tier versions have the same throttling limits. We repeated 30 trials for each of the 12 permutations of message size and edition, with each trial being run sequentially to avoid interference. Fixed parameters were 10 sensors, constant IMT of 200ms, runtime of 120 seconds.

During an initial run we exceeded the message throttling limit for the B1 edition IoT Hub. This caused the subsequent trials of larger message sizes to yield latencies with very large outliers. We re-ran this experiment and confirmed that B1, B2, and B3 all follow the same latency pattern for large message sizes. For space reasons we only include the results from B3.

Figure 1 shows the Cumulative Distribution Function (CDF) of B3 messages. 2048B and 8192B messages follow nearly the same trace, crossing the 50% probability mark at 32ms. 32,768B messages show a higher latency until the curve plateaus. The 512B messages follow the other message sizes at first when P(L) < 50%. However, between 38ms to 41ms latency, the distribution dips as a result of the high standard deviation. As latency increases though, the 512B curve joins the message sizes in plateauing so that all message size cross the 95% probability mark at less than 60ms.

Effects of Varying the Intermessage Gap Time. Sensors may send data at various frequencies ranging from thousands of times a second to once a minute. Very high send rates can overwhelm the message consumers even if throttling limits are obeyed. We designed an experiment in which data is sent to the cloud at different IMTs, holding the message size constant at 2048B, and keeping the overall message sending rate for each IMT within throttling limits. We provisioned one unit of B3 Edition IoT Hub with 4 partitions, and generated messages for about 300 seconds in each trial, sending data at three constant IMTs of 10, 100, or 1000 ms.

We repeated this experiment using the Pareto distribution for IMT. We chose shape and corresponding scale parameters that would produce median IMTs of 10, 100, and 1000 ms. For example, for shape 2, scales of 7, 71, and 707 produce messages with medians of 10, 100, and 1000 ms, respectively.

Finally, we split the experiment into three groups. Each group ran six trials per IMT, so 18 trials in total per group for the three IMTs. Group 1 had one sensor sending messages at the specified IMT, while Group 2 had 10 sensors. Group 3 had one sensor sending messages with the Pareto distribution.

For both constant and Pareto trials, the mean latency is between 31.1 and 53.9 ms. When high frequency runs (those with a median IMT of 10 ms) are removed the mean latency falls between 31.1 to 33.1 ms. The standard deviations of Pareto runs are higher than their constant counterparts but results are similar to constant trials.

For both constant and Pareto distributions, the target median IMT of 1000 ms has very few spikes in latency, and the target median IMT of 100 ms has slightly more spikes. However, with a target median IMT of 10 ms, spikes are very frequent and considerably worse in latency as seen in Fig. 2. In all figures, relative send time refers to the amount of seconds elapsed since the first kept message (after the 5% drop). For the constant distribution, these spikes have a regular pattern, but this was not the case for the Pareto distribution.

Due to the way partitions work, all messages from a single sensor go to a single partition in the underlying Event Hub. IoT Hub ensures that all messages associated with a single sensor are persisted in the order they were received. To preserve order, IoT Hub imposed hidden limits on the fastest send rate in our trials (IMT 10 ms). The processing pipeline temporarily pauses reading of new messages until it has flushed some of the existing messages to the underlying Event Hub. The end result is a short latency spike.

Effects of Varying the Partition Count. Once an IoT Hub partition number is set it cannot be changed. The partition count does not affect the cost. But, increasing the number of partitions increases the number of concurrent readers for incoming messages. We tested whether latency decreases as the partition count increases.

We created four B3 edition IoT Hubs with partition counts of 4, 8, 16, and 32. We sent messages at a constant IMT of 10 and 100 ms for each partition. Each IMT was repeated for five trials using ten clients, 2048B messages, and a 300 s duration. This is well below the 6000 messages per second throttling limit for the B3 edition of IoT Hub. However, the latency was extremely high for 4, 8 and 16 partitions. There is no load balancing between partitions within IoT Hub, and a sensor always sends to the same partition as previously stated. Thus, whenever a small number of sensors sends to a small number of partitions, there is a high likelihood that partitions will experience an unbalanced load. Our worst case scenario would be when all 10 of our sensors send to a single partition. The IoT Hubs with fewer partitions have a greater probability of higher average latencies that rise drastically over time.

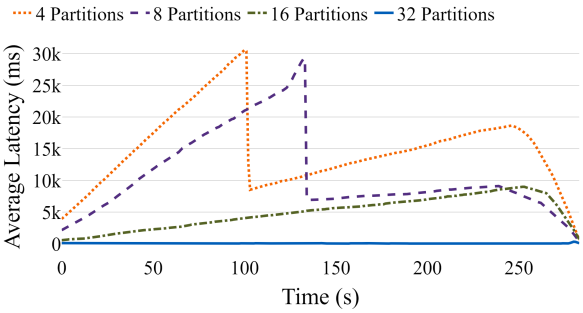

Figure 3: Latency mean of 10 sensors, constant IMT of 10 ms.

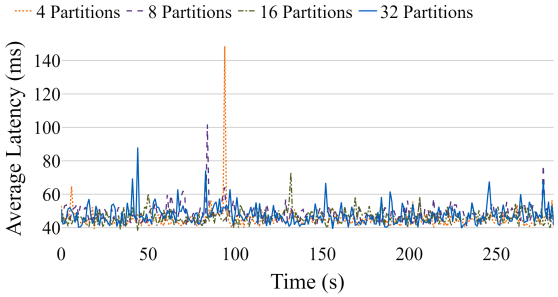

Figure 4: Average latency (ms) of 1000 sensors, constant IMT of 333 ms, varying partitions.

Sixteen partitions have a more gradual climb in latency than 4 or 8 partitions for the trial shown in Figure 3, but this was not always the case. In some trials that used different unique sensors, the 16 partition IoT Hub performed similarly or even worse than the 4 and 8 partition IoT Hubs. This can be explained by unbalanced loads, perhaps due to deterministic sensor ID hashing.

A count of 32 partitions handles this load well. 65% of the messages have a latency under 50 ms, 83% have a latency under 100 ms, and 97% of the messages have a latency under 200 ms. A possibility still exists for unbalanced sensor assignments between partitions, even with 32 partitions. However, the best performing workload for Azure IoT Hub, and the one for which the design is optimized, is one with a large number of partitions and with messages that come from a large number of sensors with a modest message frequency.

Another set of experiments tested latency characteristics across 4, 8, 16, and 32 partitions with 1000 sensors. Each sensor sent messages with IMT of 333 ms, or about 3 messages per second. Figure 4 shows that the performance of 4 partitions is poorer than higher partition counts but also that this workload is manageable for IoT Hubs across all numbers of partitions. Latency spikes are present, but they are less extreme with no evidence of latency rising over time.

With 1000 sensors across all trials, IoT Hubs with 8, 16, and 32 partitions have 80% probability of latency under 50 ms, 91% probability of latency under 100 ms, and 99% probability of latency under 200 ms. The CDF for 4 partitions follows a slightly different curve with 74% probability of latency under 50 ms, 94% probability of latency under 100 ms, and 99% probability of latency under 200 ms. These results confirm that IoT Hub is best equipped to handle a large number of sensors sending at a modest rate.

Scaling Experiments. To simulate large-scale manufacturing workloads, we tested a large number of client sensors executing on 40 core nodes and sending to Azure IoT Hub within throttling limits.

Our first trial consisted of 1000 sensors placed on 1, 2, or 4 computing nodes, each sending messages to an IoT Hub with a single B3 unit. The messages had a constant IMT of 200 ms, which was 5000 messages per second in total. This is close to, but never exceeding, the throttling limit (6000 msg/sec). All trial runs lasted 90 minutes and sent a total of 27,000,000 messages. Each CDF follows the same curve with a 66% probability of latency under 50 ms, an 85% probability of latency under 100 ms, a 95% probability of latency under 200 ms, and a 99% probability of latency under 300 ms.

We scaled up this experiment by increasing the number of receiving IoT Hub units. We ran 2000 sensors sending to 2 units of IoT Hub and 4000 sensors sending to 4 units of Iot Hub. We simulated these sensors on 1, 2, or 4 Palmetto computing nodes. During these larger trials we sent 10,000 messages per second and 20,000 messages per second respectively. We stayed below the throttling limits given for the multiple IoT Hub units. Trials continued to be run for 90 minutes. Message latency did not show any substantive change as the count of sensors was scaled up. The CDFs from each experiment all follow the same curve. Azure IoT Hub behaved as expected under these large, stable workloads, even with message send rates above 80% of its throttling limit.

5 CONCLUSIONS

Azure IoT Hub is designed to scale horizontally and achieves the best results when a large number of sensors send data at an aggregate rate within the IoT Hub's throttling limit. This benefits manufacturing plants with a vast array of sensors sending independent data streams to the cloud. However, individual sensors should avoid sending data at high frequencies to not overwhelm IoT Hub's message consumers. Adding partitions reduces latency without increasing cost, while message size is optimal at a few kB. Knowing this, it is relatively simple to determine the configuration and flat rate cost of an IoT Hub deployment based on a plant's needs. Our generator's source code, scripts, and data are all public [2].

6 ACKNOWLEDGEMENTS

This research was supported by NSF #1405767 and #1725573. We thank Robert Underwood, Brian Miller, and Jim Pepin from Clemson; and Joey Brakefield, Steve Busby, and Todd Whitehurst from Microsoft for technical contributions.

REFERENCES

[1] J. W. Anderson, K. E. Kennedy, L. B. Ngo, A. Luckow, and A. W. Apon. 2014. Synthetic data generation for the internet of things. In *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, 171–176.
[2] Amy Apon. 2020. LTB2021 Digital Plant. https://github.com/aapon00/ltb2021.
[3] A. Bahga and V. Madisetti. 2011. Synthetic Workload Generation for Cloud Computing Applications. *Journal of Software Engineering and Applications* 4, 7 (2011), 396–410. https://doi.org/10.4236/jsea.2011.47046
[4] Nicole Berdy. 2016. IoT Hub throttling and you. https://azure.microsoft.com/en-us/blog/iot-hub-throttling-and-you. https://azure.microsoft.com/en-us/blog/iot-hub-throttling-and-you
[5] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking control of the enterprise. *SIGCOMM Computer Communication Review* 37, 4 (2007), 1–12.
[6] B. Posey, A. Deer, W. Gorman, V. July, N. Kanhere, D. Speck, B. Wilson, and A. Apon. 2019. On-Demand Urgent High Performance Computing Utilizing the Google Cloud Platform. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. IEEE, 13–23. https://doi.org/10.1109/UrgentHPC49580.2019.00008
[7] B. Posey, C. Gropp, B. Wilson, B. McGeachie, S. Padhi, A. Herzog, and A. Apon. 2018. Addressing the Challenges of Executing a Massive Computational Cluster in the Cloud. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 253–262. https://doi.org/10.1109/CCGRID.2018.00040