inductive AUTOMATION 2110 21st St., Suite 500 Sacramento, CA 95818 1 (800) 266-7798 www.inductiveautomation.com

Inductive Automation

Benchmarks 2008

This whitepaper covers the details of running our products, FactoryPMI™ and FactorySQL™, through the paces of rigorous benchmark tests. These tests were designed to push the outer limits of what our software, especially our SQLTags™ system, is capable of, and to discover what those limits are. The information that follows should help answer most of the performance related questions that we commonly encounter from customers evaluating our software. In the spirit of openness that we strive for, we've made the results of these benchmark tests free and available for all.

This paper is broken into two parts. Part one tests FactoryPMI under high concurrent client load, especially with heavy SQLTags usage. Part two (starting on page 6) tests FactorySQL SQLTags throughput against different database systems.

FactoryPMI Benchmark Tests: Client Load Test using SQLTags

Goal

The goal of this benchmark test was to see how the FactoryPMI Gateway responded to increasing concurrent client load, especially under heavy SQLTags usage. Given FactoryPMI's web-launched client architecture and generous licensing, concurrent client counts can be expected to be quite high for many installations. This benchmark aims to answer the question: *"How many clients can I realistically expect to run at a time?"*

Methodology

In the absence of licensing limitations, the answer to the question posed above depends on the computing resources available, as well as the load that each client puts on the Gateway. For these reasons, we actually ran three separate load tests to determine how many concurrent clients a Gateway could support under various conditions. Note that because of the enormous number of computers needed to run a client load test of this size, the test was performed on an on-demand virtual computing platform. The performance specs below are the dedicated-equivalent specs for each virtual computer.

For these tests, we had two different size servers to run the Gateway on. The *small server* had a 1.2 GHz Xeon processor and 1.7 GB of RAM. The *large server* had 2 dual-core 1.2 GHz Xeon processors and 7.5 GB of RAM. The small server was running Ubuntu Linux 7.10 32-bit edition, and the large server was running Ubuntu Linux 7.10 64-bit edition. Both servers were running MySQL 5 as the database, with a SQLTags simulator driver.

We also had two different size projects. When we talk about the "size" of the project, in this context, we mean the number of tags that are subscribed at once. The total number of tags in the system is irrelevant to this test, only the number of tags being processed by each client affects the Gateway's performance as the number of clients scale up. Our *small project* monitored 30 tags at once. Our *large project* monitored 300 tags at once. These tags were all in one scan class which ran at 1.5 seconds. 75% of the tags were changing every scan. This means that a client subscribed to 300 tags was processing 150 tag changes per second.

The three tests that we ran were:

1. **BM1**: Small Server / Small Project.
2. **BM2**: Small Server / Large Project.
3. **BM3**: Large Server / Large project.

For each test, we took six measurements as we scaled up the number of concurrent clients. The measurements were:

1. **End-to-End Write Response.** We measured the amount of time between issuing a SQLTag write and seeing the written value come back through a subscription to that tag. With our architecture, this measures the worst-case path through our various polling mechanisms. The SQLTag simulator driver also used a 50ms write penalty to simulate the cost of writing to a PLC over ethernet. The tag that we wrote to was in a special fast scan class of 250ms. Results shown are the average of 4 write/response roundtrips.
2. **CPU.** The overall CPU load for the server.
3. **RAM.** The amount of RAM that the FactoryPMI Gateway and MySQL database were using.
4. **Network I/O.** The total throughput of data through the network card. We had some trouble measuring this as load got high on the small server, because the computational overhead of the application (iptraf) that we used to measure the throughput adversely *affected* the throughput when it was turned on. The results are still useful however, and show a linear increase in throughput until the measurement tool starts failing.
5. **SQLTags Scan Efficiency.** This number is a percentage that represents the Gateway's (Actual Scans / Second) / (Ideal Scans/Second) for the SQLTags provider that we were using. With a delay between scans of 200ms, the ideal scans/second is 5. This number is a good indicator of how much strain the Gateway is under.
6. **Client Request Efficiency.** This number is a percentage calculated as follows: (Total Client SQLTags Requests/Sec) / (# Of Clients * Ideal Client Requests/Sec)**.** For example, with a delay of 250ms, the ideal rate for clients to poll is 4 times per second. If you have 5 clients running, and your total client poll rate is 18 requests/sec, your efficiency is 18 / (4*5) = 90%. This number is also a good indicator of Gateway strain, the less efficient it is running, the slower the clients receive updates.

Data

We ran our client load benchmark tests up to 150 concurrent clients. You may notice that BM2 (Small Server / Large Project) stops at 130 concurrent clients, because at that level its performance was unacceptable. BM1 and BM2 both made it all the way to 150 clients.

First we'll look at the numbers for the End-to-End Write Response, abbreviated E2EW. We feel that this number is most intuitive for system performance. This number represents the delay between pushing a button or changing a set-point, and seeing that change come back to the client's screen through the PLC. Because this operation goes through every part of SQLTags, any portion of the system that was being adversely affected by load stress would affect this time. Of course, lower E2EW times are better. We strive for sub-second E2EW as a rule-of-thumb for a fast, responsive HMI.

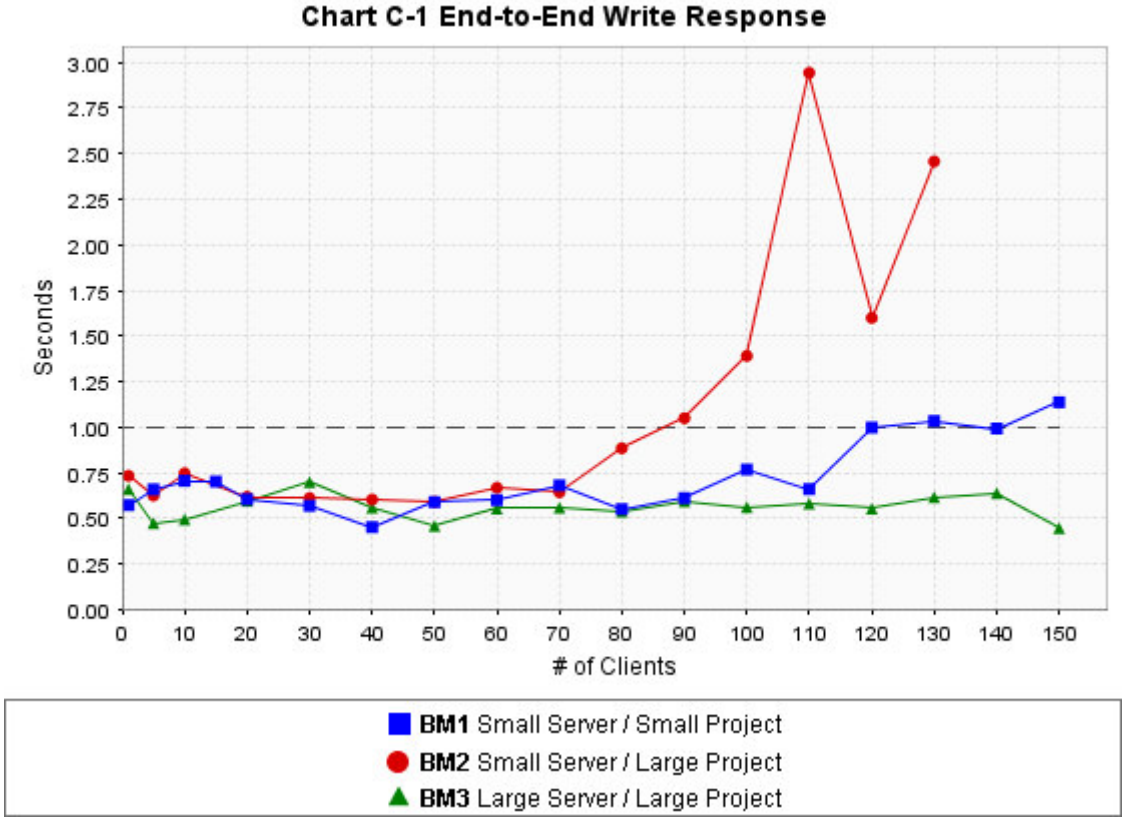
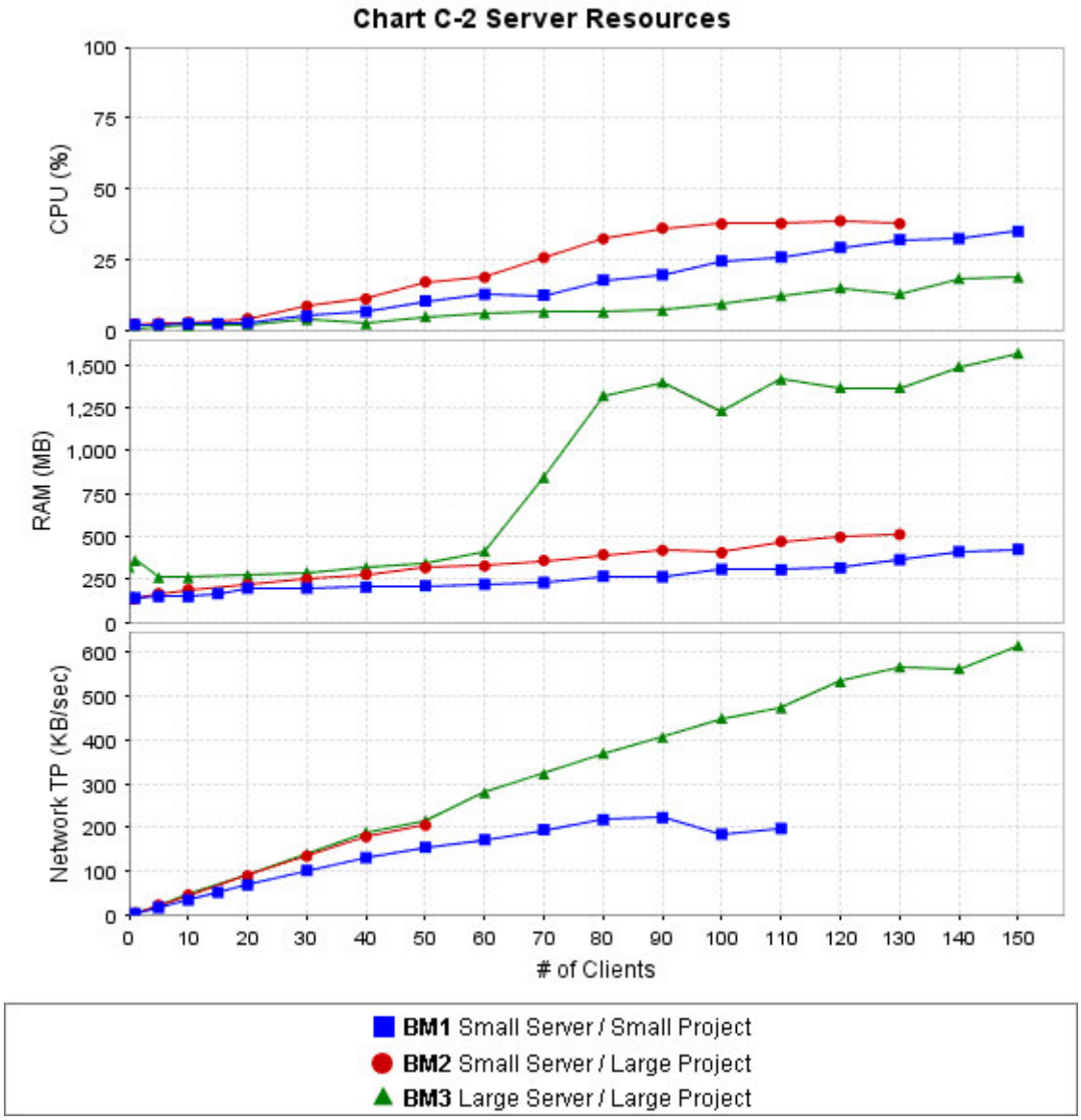


Chart C-1 shows the results for the E2EW test on all three load-test benchmarks. All three scenarios maintain sub-second E2EW times until 80 concurrent clients. After this, you can see that BM2 starts to suffer degradation in performance. BM1 starts to slow down at higher numbers, and not as fast as BM2. BM3 maintains even E2EW times all the way through 150 concurrent clients.

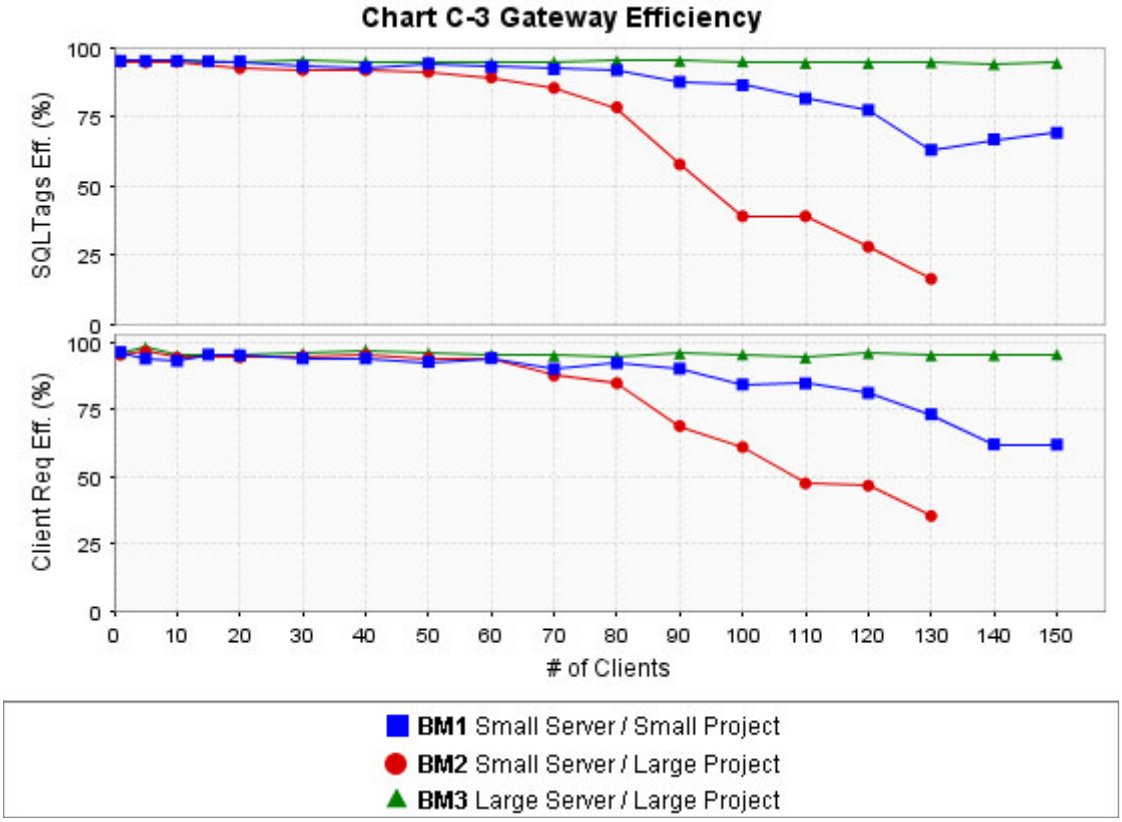
Next we'll look at the resource usage for the computer running the FactoryPMI Gateway and the database. These numbers are a good gauge for how hard the server machine is working.



As you can see in Chart C-2, all three tests show moderate CPU usage, and slow linear increase. This shows that raw CPU power wasn't the limiting reagent. Even as BM2 hit a performance wall around 80 clients as we saw in Chart C-1, its CPU usage did not spike.

Ram usage is a similar story, except for BM3, whose RAM usage spikes around 70 clients. We're not entirely sure why this is, but one major difference between BM3 and the other two tests was that BM3 was run on a *server VM*. Java has two flavors of virtual machine – client and server. The server VM is less shy about grabbing resources as it wants them, because it knows it is server software. The machine that BM3 was running on had 7.5 GB of RAM, so there was plenty to go around. Judging by the RAM numbers, this was not what caused performance slowdowns at higher numbers either.

Network Throughput is another story. As we mentioned above, the software that we used to measure network performance was unable to measure accurately once the load got high on the smaller server. However, we can see that the network throughput increases linearly as the number of clients go up, as we would expect. These numbers suggest that our small project required about 3.5Kbps (that's kilobytes, not kilobits) per client, and our large project required about 4.5Kbps per project. It seems that this was the limiting factor for the smaller server, which had less networking resources available to it.



Lastly, Chart C-3 shows the gateway scan efficiency. These numbers corroborate what the E2EW numbers told us. We see that BM2 hit a performance wall around 80 concurrent clients. Above this level, BM2's SQLTags scan efficiency is severely affected. What this means is that the Gateway was unable to poll the SQLTags system very quickly. Instead of 5 polls per second, at ~20% efficient (the last datapoint), it was only able to poll once per second. These scan rate slowdowns are what caused the E2EW times to increase. Also notice that BM3's scan efficiencies never falter, which is why the E2EW times never increased.

Conclusion

We were very pleased with the results of these benchmarks. Our average customer launches in the realm of 5 – 25 concurrent clients, meaning they aren't even approaching stress levels for the Gateway. This benchmark shows that much larger

installations are well within the realm of reason. We were very pleased with the grace that the SQLTags system scaled, which is precisely what it was designed for. These benchmarks validate the efficacy of our web-launched, database centric architecture.

FactorySQL Benchmark Tests: SQLTags

Goal

FactorySQL, as the central data processor of Inductive Automation's software architecture, is responsible for tying together OPC data with standard SQL databases. The introduction of SQLTags technology has reduced the visibility of the central database, without reducing its role. In discussing the software architecture as a whole, it is common to ask "how many tags can the system have?" Given the database centric architecture, the answer to this question has multiple dependencies: FactorySQL, FactoryPMI, and perhaps most important, the performance of the database system. Given the vast range of variables that are introduced when designing a system- hardware configurations, database configurations, and project configurations to name only a few- it would be very difficult to provide specific numbers indicating what one might accomplish. It was therefore the goal of the benchmark tests to instead provide a reliable guideline that could be taken at face value to quickly identify the viability of a configuration.

Methodology

When discussing the question of "how many tags the system can handle", there are really two separate questions being discussed: How many tags can be configured in the overall system, and how many tags can be reliably serviced according to their configuration. Put a different way, the later question asks "what is the tag throughput?" while the former asks "will simply adding more tags to the system decrease that throughput?"

Two primary tests were conducted to explore these questions. In the first test, a relatively small fixed number of tags were created. Each tag was set to update on every execution cycle, when enabled. The number of enabled tags ramped up over time, while the average execution duration of the scan class(es) was recorded. This test illustrated the raw performance of each database system.

In the second test, two variables were modified in order to maintain a specific amount of work. The first variable was the percentage of the tags changing per cycle. Second to this was the number of tags in the overall system (with all tags active). The test attempted to reach a sustained 1 second average execution duration for each of the

scan classes involved. This test demonstrates how throughput is affected by the overall number of tags in the system.

In both of the test situations, multiple trials were conducted with two different configurations. In the first configuration, all tags were executed under a single scan class. In the second configuration, each tag type was executed in its own scan class. In both configurations tag types were equal ratios: 1/3 integer, 1/3 floating point, and 1/3 boolean. These additional constraints were designed to indicate a particular efficiency (or lack thereof) with a particular data type, as well as to explore the possible benefit of executing the tags in a multiple threads (as multiple scan classes would benefit from FactorySQL's heavily multi-threaded architecture).

All tests were conducted on a medium range PC (Dual core 2.16 GHz processors, 2 GB ram), with base installations of the databases. It is obvious that improved results may be possible with increased hardware capabilities and attention to database tuning. However, this configuration was chosen given the wide variability in these options, the goals of this benchmark, and the observed pattern of usage among our customers.

The following database systems were tested (they will be referred to henceforth by the provided abbreviation):

- **DB1:** MySQL, with MyISAM data engine
- **DB2:** MySQL, with InnoDB engine
- **DB3:** Microsoft SQLServer 2005 Express
- **DB4:** Oracle 10g Express

Data – Test 2a

In the first test set, 18000 tags were created in each database. Data types were evenly divided in thirds: 6000 integers, 6000 floats, and 6000 booleans. All scan classes were set to execute at a 1 second rate, and each tag was set to change randomly at the same rate, ensuring a 100% tag update per cycle.

In the first test sample, all tags were assigned to the same scan class. At each tag count level samples were taken for one minute, and the average scan class execution duration was recorded.

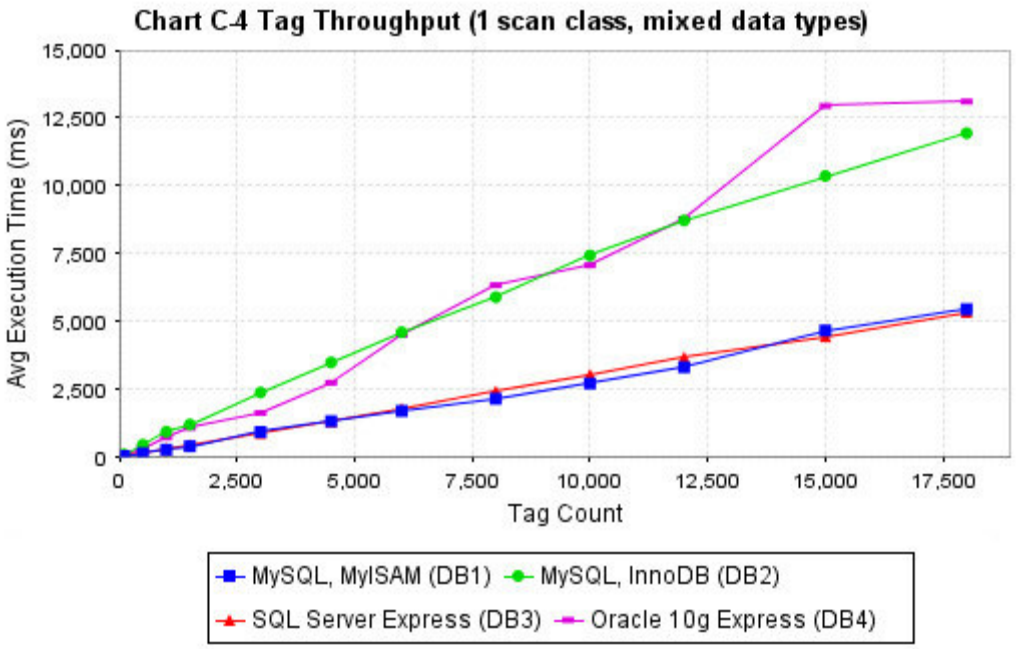


Chart C-4 shows the results of the first sample set. Each database system demonstrated a linear relationship between tag count and execution duration. In the chart above, a lower sloping line indicates more efficient performance, and a quicker execution rate. In general terms, we see an immediate similarity between DB1 and DB3, and likewise between DB2 and DB4.

Given that the scan class was configured to execute at a one second update rate, it would be natural to examine the data in hopes of finding the maximum number of tags that could reliably executed at this rate. We often refer to this as a measurement of *efficiency*, looking for the maximum throughput that would maintain 100% efficiency. To this end, the following values were observed:

Database	Tag Level	Avg. Duration
DB1	3000	918
DB2	1000	922
DB3	3000	883
DB4	1500	1009

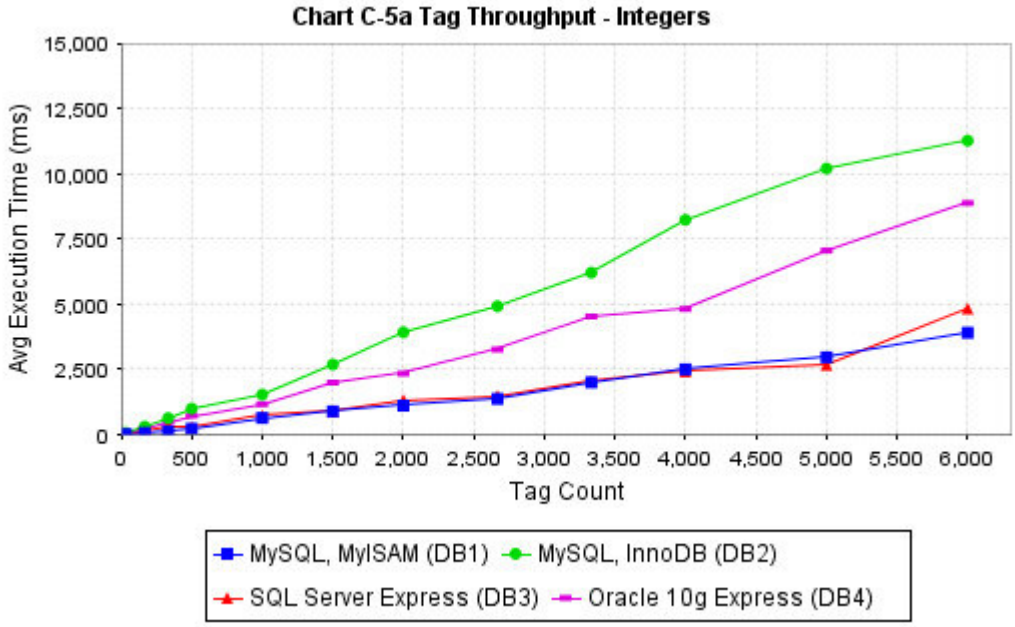
Since performance was measured for a series of tag levels, the level closest to 100% efficiency is provided in the table above, along with the precise duration value for that level.

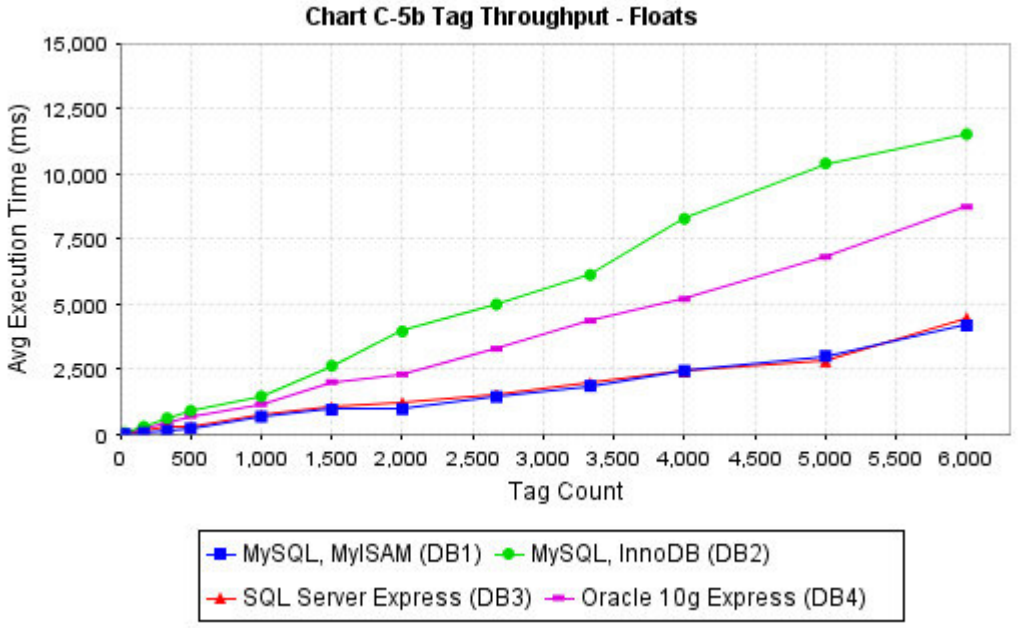
Data – Test 2b

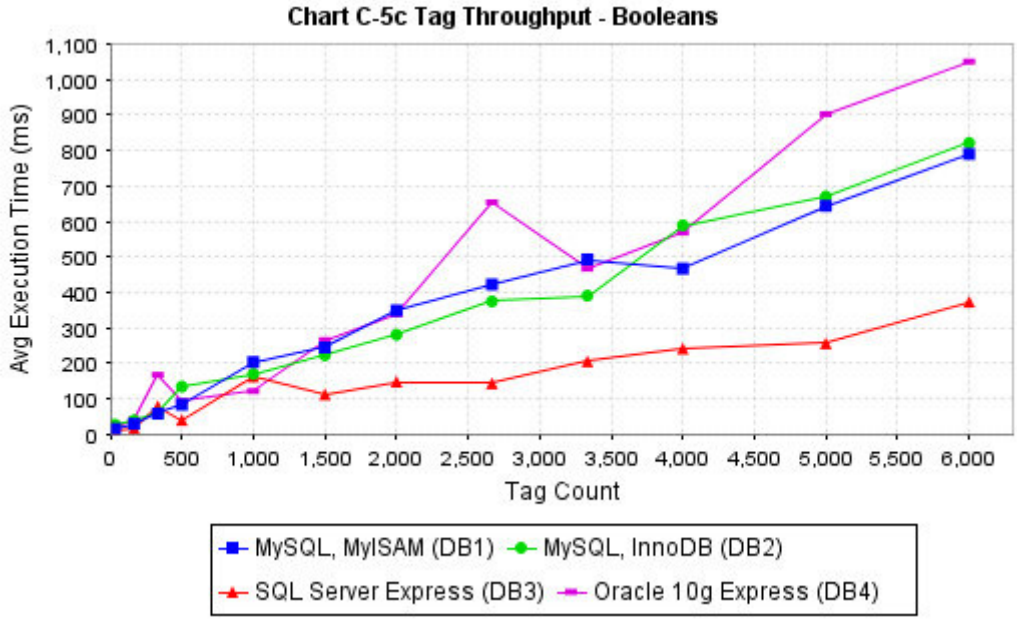
The next series of tests examined how performance was affected by the distribution of tags amongst several scan classes, based on data type. The goal of testing this configuration was multifaceted. First and foremost, due to FactorySQL's multi-threaded architecture, the inclusion of multiple scan classes causes the work to be split into parallel tasks. We were able to observe how this partitioning affected the database and the throughput that we were able to achieve. Secondly, by separating the tags into scan classes based on data type, we were able to directly identify and measure the benefit of FactorySQL's internal write operations on a per data type basis.

The same 18000 tags from the previous test were assigned to one of three scan classes based on type. Therefore, there were 6000 tags in each scan class. Each scan class was set to execute at rate of 1 second, and the tags were set to change on every update interval. Therefore all tags would be written on each execution. Other test parameters remained as they were for test 2a.

The following charts show the comparative performance of each database system versus the number of tags being executed, for each data type. NOTE: It is important to recognize the change in Y-axis scale for Booleans.







The observed results are generally in line with test 2a. Although the data is presented in three charts, it is important to recognize that all three were scan classes were being executed concurrently. We can therefore create a table similar to that of test 2a, indicating a general 1 second throughput, by adding together the number of tags obtaining approximately 100% efficiency in each class.

Database	Tag Level	Avg. Durations (int, float, bool)
DB1	13500	882, 957, 245
DB2	4500	890, 837, 89
DB3	13500	901, 1039, 113
DB4	4500	713, 723, 94

Conclusion – Test 2

The data obtained in the two sub tests of Test 2 provided several very good insights. While we were slightly under whelmed by the raw numbers of test 2a, we were encouraged to see that separating tags into multiple scan classes had a significant effect on throughput. Furthermore, test 2b indicates a strong efficiency in writing Boolean data, which would indicate that much higher tag counts could easily be achieved when using more of this type of data. The enhanced performance of this type of data is explained by two causes working in concordance: firstly, given either a true or false value, there is a 50% chance that a "randomly" selected value will be the same as the current one, absolving the need to write the value to the database. Secondly, FactorySQL optimizes database writes to compensate for reoccurring data, which naturally occurs frequently in Boolean data.

Perhaps the most important concluding notation that we can make for test 2 is to reiterate the fact that all measurements involved 100% of the tags changing value on each evaluation cycle. In a real world situation, only a percentage of the overall system changes in a given timeframe, and on the order magnitude to which we presented data in the tables above (1 second), this percentage is generally low. In test 3 we will develop this sentiment further.

Introduction – Test 3

Continuing with the conclusion of test 2, in this test we sought to identify how the sheer number of tags configured in the system would impact throughput to the database. Given our "unlimited tags" licensing, it is common to receive questions regarding the *practical* tag limit of the system, meaning the number of tags that can be effectively processed in a given time frame.

As previously mentioned, there are a wide variety of variables that makes answering such a question very difficult. For instance, what is the time frame? In a real world application, each piece of data has its own relevant update rate. The speed of a motor may change relatively quickly, on the order of milliseconds, whereas it may only be necessary to track the ambient temperature on the order of minutes. A well designed project would accommodate this scenario with a number of scan classes, preventing tags from being evaluated more than necessary. Along the same line is the discussion

of 'tag update percentage', meaning the percentage of tags that actually change value in a given cycle. This point was touched on in the conclusion of test 2. If a tag's value doesn't change, there is very little overhead in dealing with that tag. Therefore, it is expected that a system with a relatively small number of changes per cycle would be able to easily accommodate a larger number of tags.

Test 3 is divided in a manner similar to test 2: first the test is performed with a single scan class with an even mix of data types. Then, it is performed again with each data type in its own scan class. In both cases the general procedure is the same: Select a specific *tag update percentage*, and measure how many tags can be added to the system while maintaining 100% execution efficiency. Three update percentages were tested: 1%, 50%, and 100%.

Data – Test 3a

In this test there was 1 scan class executing at a 1 second update rate. Integer, Boolean and Float data tags were added in equal proportions at each update percentage level until the average execution duration was approximately one second. The results are displayed in tabular format as an effective graph is difficult, due to the range of data the axes would need to accommodate.

Table 1 – Tags in system at given update level

Update % Level	DB1	DB2	DB3	DB4
1%	218,000	90,000	158,000	130,000
50%	7,640	2,500	5,720	3,500
100%	5,000	1,300	3,165	1,600

This table shows the number of tags in the system running at 100% efficiency, with the specified update level. To get to the core of determining how a larger number of tags affects base throughput, it is useful to display the throughput at each level (Example: if 1% of 218000 are changing, throughput is 2180 tags per second).

Table 2 – Throughput (tags per second) at each update level

Update % Level	DB1	DB2	DB3	DB4
1%	2,180	900	1,580	1,300
50%	3,820	1,250	2,860	1,750
100%	5,000	1,300	3,165	1,600

These values are displayed along with their relative percent change, in order to help illustrate the degree of change introduced by the volume of tags. As in previous tests, DB1 and DB3 tended to be somewhat similar in their behavior, as did DB2 and DB4. The observed throughputs are generally in line with the results of Test 2a.

Data - Test 3b

Similar to the difference between test 2a and 2b, in test 3b the procedure was the same as the previous test, with the exception that each of the three data types were placed in their own scan class. Although data was observed for each data type, the goal of the test was to observe overall throughput. Therefore, "maximum efficiency" was determined based on the worst performance scan class. In other words, the data was recorded when one of the three scan classes reached its maximum throughput.

Table 3 - Tags in system, 3 scan classes

Update % Level	DB1	DB2	DB3	DB4
1%	277,650	170,000	281,500	130,000
50%	14,000	4,300	11,460	6,000
100%	8,000	2,000	6,000	3,200

In terms of raw throughput:

Table 4 – Throughput (tags per second) at each update level, 3 scan classes

Update % Level	DB1	DB2	DB3	DB4
1%	2,776	1,700	2,815	1,300
50%	7,000	2,150	5,730	3,000
100%	8,000	2,000	6,000	3,200

And as a percentage improvement over test 3a, with a single scan class:

Table 5 - % Improvement with 3 scan classes over 1 scan class

Update % Level	DB1	DB2	DB3	DB4
1%	27%	88%	78%	0%
50%	83%	72%	100%	71%
100%	60%	54%	90%	100%

We therefore see once again a strong improvement by using multiple scan classes. Our improvement in the 1% range is not always as large as one might expect. In fact, with DB4 we see no noticeable improvement. This can possibly be explained by a "wall" hit in RAM available on the system. During these tests, the CPU of FactorySQL remained reliably consistent, around 10%. The databases varied considerably, but generally spiked at a max of 40% (non-sustained). Memory usage, however, seemed to become a factor at the higher tag levels, with both FactorySQL and the database requiring considerable memory.

It is worth mentioning that in all of these samples the scan class with the Boolean data was executing much more efficiently than the other scan classes, as was observed and discussed in Test 2.

Conclusion

The data observed in Test 3 was generally in line with that of Test 2. We were pleased to see that we were able to maintain a high level of throughput even with a large number of tags. The similar performance of certain databases (DB1 & DB3, and DB2 & DB4) was observed once again, though certainly much less clearly. In particular DB4 appeared to provide a more consistent throughput across the middle range.

We have attempted to test the most extreme ranges to provide a framework for evaluation. We were pleased at the patterns that emerged: the strong performance of the database systems, the increased efficiency of multiple scan classes, and the very efficient handling of repetitious data. We believe the numbers indicate that a well designed real-world project (which generally tends to have a relatively low tag update percentage) could viably reach a rather large scope.

We have deliberately restrained comparative judgment on the database systems tested, aside from commenting in the similarities between them. The data clearly shows a disparity among the systems, but it's important to recognize a wide range of peripheral factors that one might consider before choosing a system. These factors are beyond the scope of this paper, but tend to include topics like corporate standards, tool availability, data integrity mechanisms, and of course, price.

Finally, we would like to reiterate that the data provided in this paper relates specifically to the SQLTags portion of FactorySQL. Though they may provide insight into the general performance of each database system, it would be wildly inaccurate to assume they applied equally to standard FactorySQL execution. FactorySQL provides a wide range of features that interact with the database through diverse mechanisms, and each mechanism may provide different capabilities. The reader is encouraged to investigate independently each feature that would apply best to their situation.

About Inductive Automation
Inductive Automation pioneered the first full-featured web-launched HMI/SCADA system in the world. Its standards based, database-centric architecture receives accolades from plant managers, IT managers, and system integrators worldwide. With a commitment to software quality and technical support second to none, Inductive Automation is at the forefront of industrial software.

