TSI_PerformanceTesting Quick Guide

Before you start

Currently, test bench supports ABB's cpmPlus History, InfluxDB, and TimescaleDB. To run the test bench to them only necessary databases/measurements/hypertables and credentials should be created to the database under test before usage. Check the first *Hint!* for more information.

The tested database should be installed and configurated on Hyper-V virtual machine before testing. Also, client implementation needs to be done on test bench source code. The client can be implemented by creating a new class which inherits IAdapter interface. By creating the body for the IAdapter functions the client is integrated into the test bench. All database specific functions can be implemented in that class also. The constructor of the class is usually used for the connection creation (and all necessary actions, e.g., getting the ids of the variables). PrepareWriteData/PrepareReadData functions are used to create the query string or getting the ids of the used variables. Those functions can be empty if the client's send function does not require any pre-processing/initialization. Send/Read functions should contain as minimal functionality as possible for writing/reading. After the class is ready the class constructor should be added to the Main function where database type is selected.

Database credentials (username, password, database name, table name) should also be provided in the source code as there can a lot of differences between the databases on how/what should be provided.

Hint! *By default, cpmPlus History client is configured to use username: rtdbadmin, password: mypw. Also, the guest user is used during the testing, so it should have read permissions to the variables. For InfluxDB by default database:mydb, user&password: admin, second user&password: admin2, and measurements: data. For TimescaleDB database name is tutorial, user: rtdbadmin, password: mypw, hypertable name: data. These should be created into the database before testing.*

The used data schema is (timestamp, name, status, value). As the data type support can differ among databases the test developer should decide the best data type for each field. Default datatypes for the fields are (microsecond precise timestamp, string, string, int32). Data schema with required tables/other data formats should be configurated to the database before testing.

Usage

Test bench can be run from the command line and the only required command line parameter is the path to the configuration file. Example configuration file is included with this documentation. The configuration file uses .ini file format. Controlling of the test bench is done via the configuration file. With Ctrl+C running test can be stopped if needed and measured results are saved before closing.

Results

After the test bench is finished the results are saved into the current user's document folder under TSI folder (e.g. C:\Users\admin\Documents\TSI\..). There each test run has its own folder and under test, each test type that has been run have own results folder where the measured values are on their own

raw values .txt files. These files are tab separated and they can be imported to some data analytics program, e.g., Microsoft Excel.

Hint! *Cpu_metrics file contains CPU usage(percentage) of each core on their own column*

Future development

Currently, the test bench uses random values for each variable. To change this to some other type of data (sin wave) new value creation which is in either PrepareWriteData or Send function (depending on the database API's send function type) in each database's own client class.

TSI_PerformanceTesting Detailed Documentation

Here the developed test bench' design and implementation are presented. First, the overall architecture of the system is examined. Then the main components of the system: data production, data consumption, and telemetry recording are presented in more detail.

1.1. System architectural overview

The test bench is written on C#, and it has a configuration file that is used to control the test parameters. A tested database is deployed on a virtual machine as a single node solution. The test bench does not automate virtual machine deployment or database configuration that must be done before the test bench is run. As it was seen in the previous chapters and related work, the test bench should support different scenarios. The test bench has three different test cases which try to simulate three common IIoT use cases which are:

- On-premise IoT, heavy data insertion with random reads
- PIMS/cloud-based, random inserts with heavy reading
- Monitoring, concurrent data insertion, and reading

With these use cases, the test bench tries to take IIoT point-of-view to the performance measuring. On-premise IoT scenario is a smart factory type scenario where the raw data is sent to a local database, and from there more aggregated values are sent in batches to another database. In this scenario database load is more data write-oriented and the write performance of the database is more critical. PIMS/cloud-based scenario would be another database where the aggregated values are sent from the premise. The database is under heavy read-type load as the data is used for more analytical tasks and the read performance of the database is more critical in this scenario. The third scenario simulates on-premise monitoring scenario where the raw data is under real-time monitoring and the combined performance of read and write operations of the database is crucial. Figure 1 presents the overall architecture of the system.

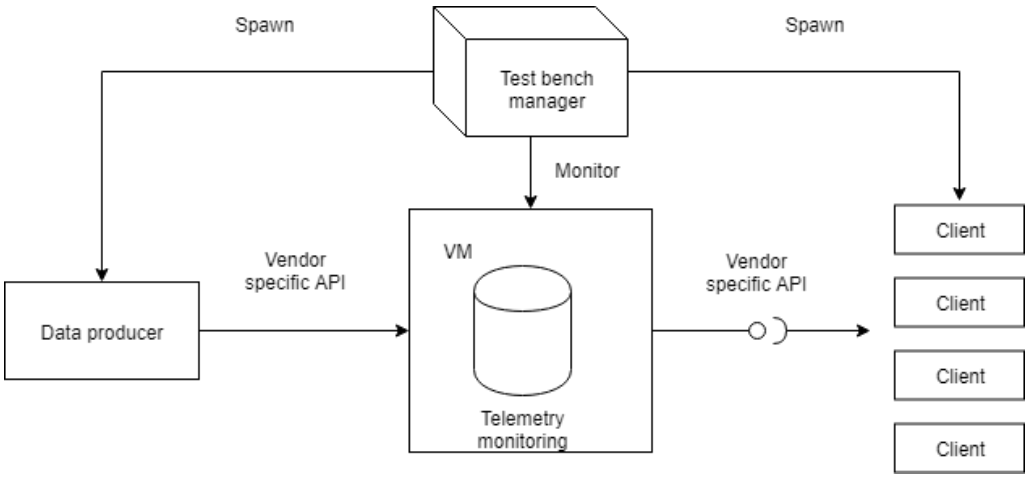

Figure 1. System architecture.

Test bench manager maintains the test runs and spawns the necessary data producer and clients who read the data. Test bench manager also spawns the telemetry monitoring to the virtual machine that holds the database under the test.
The first test case tests the write performance of the database with measurements as the number of data points to write per second and the delay of when the written data is available for queries. The second test case tests the read performance of the database with measuring the time it takes to complete the query. Testcase has three different subcases where timespan, the number of variables, and the number of connected clients is increased. The third case tests the effect of reading the concurrently written data and how it affects on write and read performance of the database. All test cases can be run separately or in sequence. From the earlier identified requirements, the test bench is targeted for scalability, performance, resource consumption, and lifecycle. Other requirements (e.g., redundancy, API support, cost) are evaluated based on documentation as they are hard or impossible to measure in the test bench.
As the database's performance might differ over time because of disk saturation, the test bench is designed to support long test cases. In these long test runs, test bench writes data to the database and starts to measure the database's performance after a given time, e.g., one week. This way the real performance of the database can be measured. Also, the lifecycle management that was one requirement for the database can be tested with longer test runs.
The test bench is modular enabling testing of a new database by only implementing the read and write functions that are typically vendor specific. Also, the virtual machine that has the database should be created.
The number of variables to write and write frequency, number of variables and timespan to read can all be modified for different workload scenarios, the test bench allows the user to run the test bench for user's specific scenario. This design differs from earlier implementations which use some specific dataset, and it could not be modified.
Test bench saves all collected measurements on own text files which use tab separation format. Each test, subtest, and telemetry collection measurements have their own recording file. Test bench doesn't implement result processing, but it is left for dedicated programs, e.g., Microsoft Excel.

1.2. Data production

Data production is done by using vendor-specific API and especially the API that is recommended for best performance. Produced data is time series type of data where the datapoint has time, value, and status. Produced data also has a variable name which represents a measurement, e.g., the rotation speed of a motor. Hence the produced datapoint has four columns as

$$X = (timestamp, variable\_name, status, value),$$

where datatypes are timestamp, text, text, and integer. Data is written into a single table or another similar type of data structure to keep the databases comparable as their data structure can vary. In the write performance test case the number of variables is increased after configurated iterations. Iterations define how many measurements of write performance is taken with one variable count. During each iteration, a new random value is inserted for variables. The frequency of write operations can be configurated for different types of write-loads. Write performance is measured as an average time it took to write the specified number of data points. Depending on the write operator's behaviour the execution time is either the write operation(s) duration or the read-back duration with a threshold value. As the default write frequency is one per second, the number of written variables is equal to the number of written datapoints. The most recent written value in each write iteration is read back

to verify a successful write operation. Value is also read back to measure the delay that the data has until it is available for reading once it has been written. Each iteration duration is fixed for one second. If the write frequency is more than once per second, only the last write operation of that iteration is verified. Hence write is verified only once per second. Data production can be configurated with initial iterations where data is written to the database without performance measuring to fill the database. By filling the database with initial data, the average on-production performance can be measured.

Measured metrics, write operation duration and duration to read the value back, are standard and hence valid benchmarking metrics. Metrics are dependable only on the number of written values and used communication protocol. As some databases only offer one interface to it the used communication protocol and part of databases overall performance. Duration to write the same number of values should also be repeatable. If the values differ then it shows that the database's performance is not static. Duration should also be linear when the number of written values is changed linearly.

1.3. Data consumption

Data read test case has three different subcases where timespan of one variable, number of variables, or number of connected clients are varied. The timespan variations measure the read performance of the database over time field, and it measures how many data points can be read from the database per second. Testcase measures how time increase in timespan and hence the number of queried datapoints affects on query completion time. As the key query in time series databases is against time field of the datapoint, this performance is key read performance type of the database. The second subcase measures how the database performs when multiple variables are read from the database. As in the IIoT environment, one device can send multiple measurements which correspond to multiple variables, read performance over multiple variables can be important for cases that kind of devices are monitored. In this test case, the timespan of each variable is static and can be configurated. The third subcase measures how the number of reading clients affect on query competition time of one client. In this test case, other clients query a configurable number of random variables on given timespan. As seen in the requirements the database should be scalable as there can be multiple devices trying to write and read to the database. Hence the performance of multiple connections should be measured.
Read query tries to read all datapoint fields and the search query is targeted with variable name and time fields. Figure 2 presents an example search query of multiple variables, but the query can differ because of database own query languages. The result of read queries is also verified that all queried data was received, and the correct amount of data points was retrieved. To be sure of the amount of data that the database has before read tests, data is written to the database with fixed timestamps and data is queried against those timestamps.

*Select * from data where variable_name in ('testinstance_1','testinstance_2') and time > now() – interval '5 min';*

Figure 2. Example search query.

As measured metrics are time-based, they are also valid performance metrics. Read operation duration can differ based on how the used drive is fragmented. Fragmentation can be compensated with long test runs. Also, the more static the duration is, the better performance the database has. Hence the linearity of the metric indicates worse performance for the database. When comparing different databases, results should be comparable then same hardware and number of values are used.

1.4. Telemetry recording on resource consumption

Telemetry recording measures the resource consumption of the database. Measured metrics are CPU usage, RAM usage, Disk usage, Disk space, and Network usage. As the virtual machines are Microsoft Hyper-V virtual machines, the telemetry recording is done via Powershell. Powershell gives access to Hyper-V statistics which record the resource usage of each virtual machine. In the write test, resource consumption statistics are saved before each variable increment so the effect of write load increment to the resource consumption can be examined.

Similarly in the write test, before each increment in timespan, variable count or client count the resource consumption statistics are saved. As the disk space of the virtual machine is measured before and after the test case, the compression of the database can be evaluated with the difference of those measurements. Network usage can also be crucial in some situations and measuring that gives insights how network depended the database is.

Example configuration file

```
[Default]
VMName="Ubuntu16 InfluxDV"
Database="InfluxDB"
DomainName="DESKTOP-0H22065"
Tests="Write"                                   ; Use | to include multiple tests. Write|Read|Mix|All

[WriteTest]
Variables=15000                                 ; Number of written values
Iterations=0                                    ; Number of measured write iterations for each variable
count
UpdateFrequency=10                              ; Write frequency (updates/second)
Increament=1                                    ; How much number of written variables is increased after
Iterations
StopValue=15000                                 ; Number of variables when test stops
InitIterations=0.01:00:00                       ; How long data is written before measuring
InitVariables=15000                             ; How many variables is

[ReadTest]
WriteFrequency=100                              ; Write frequency of test data (updates/second)
FillTime=0.00:00:00                             ; How long other fill data is written after test data
TimeSpanInitValue=0.00:01:00            ; Initial timespan that one variable is read
MaxTimeSpan=0.00:17:00                          ; Maximum timespan that one variable is read
TimeSpanIncreament=0.00:16:00                   ; How much timespan is increased after TimeSpanIterations
TimeSpanIterations=2                            ; Number of measurements taken with certain timespan read
VariablesInitValue=5                            ; Initial number of variables for Variable read test
MaxVariables=1011                               ; Maximum number of variables for Variable read test
VariableTimeSpan=0.00:01:00                     ; Timespan that variables is read in the Variable read test
VariableIncreament=1000                         ; Number of variables that is increased after VariableIterations
VariableIterations=2                            ; Number of measurements taken with certain variable count
ClientsInitValue=5                              ; Initial number of clients for Client read test
MaxClients=116                                  ; Maximum number of clients for Client read test
ClientIncreament=1                              ; Number of clients that is increased after ClientIterations
ClientTimeSpan=0.00:02:00                       ; Timespan that clients are reading the values from
ClientVariables=5                               ; Number of variables that the clients are reading
ClientIterations=20                             ; Number of measurements taken with certain client count
```

```
[MixTest]
InitIterations=0                    ; Number of seconds before measuring starts
WriteFrequency=10                   ; Write frequency (updates/second)
WriteVariables=15000                ; Number of written variables
WriteIncreament=1                   ; Number of incresed written variables
WriteMaxVar=15001                   ; Maximum number of written variables
ReadVariables=10                    ; Number of read variables
Iterations=18000                    ; Number of measurements taken
TimeSpan=0.00:05:00                 ; Timespan of read operations
```