ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA056

TANGO CONTROLS BENCHMARKING SUITE*

M. Liszcz, P.P. Goryl, S2Innovation, Kraków, Poland

Abstract

Tango Controls is a client-server framework used to build distributed control systems. It is applied at small installations with few clients and servers as well as at large laboratories running hundreds of servers talking to thousands of devices with hundreds of concurrent client applications. A Tango Controls benchmarking suite has been developed. It allows testing of several features of Tango Controls for efficiency. The tool can be used to check the impact of new developments in the framework as well as the impact of specific network-server and deployment architecture implemented at a facility. The tool will be presented along with some benchmark results.

INTRODUCTION

Tango Controls [1] can be used at both small and very large laboratories and scientific facilities. The efficiency, performance and resource utilization are important qualities of any control system as they allow for building large and complex installations which scale up to thousands of devices and hundreds of clients. Deploying Tango at large scale requires solutions for efficient monitoring and evaluation. Ability to assess the performance, detect regressions and identify bottlenecks is important for developers who work directly on the Tango Kernel as well as for users and for administrators who deploy Tango at their institutes. The developers need immediate feedback on whether the changes they made to the source code impact the performance. The administrators need to know how their hardware and network infrastructure affect the overall efficiency of the system. The users might be interested if the performance of Tango improves after an upgrade to a new version.

To address the need for efficiency monitoring in automated and controlled way, the Tango Controls Benchmarking Suite was developed.

BENCHMARKING SUITE

The Benchmarking Suite is a set of tools that facilitate the process of evaluating how Tango efficiency is impacted by qualities like the number of connected clients or the number of devices hosted in a device server. The suite consists of: a set of benchmark scripts, three device servers called *target device servers* and a benchmark runner script. A detailed description of each component is provided later in this section. A diagram showing all the components and relations between them is is depicted in Fig. 1

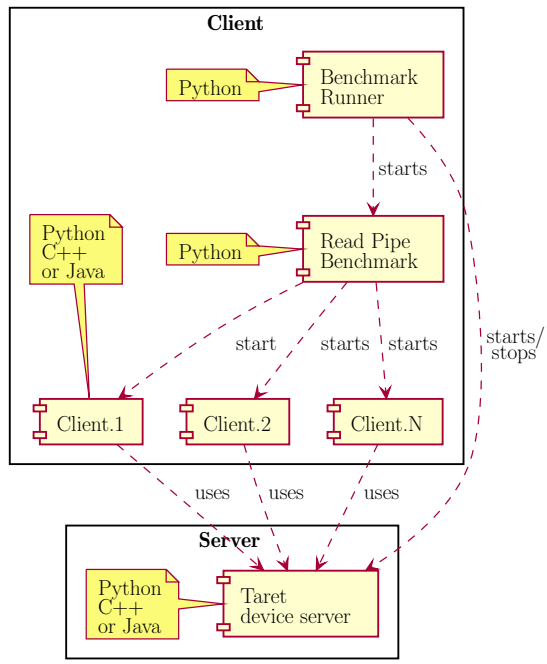

Figure 1: Components of the Benchmarking Suite.

The Benchmarking Suite was designed to be extensible. This allows users to easily write their own benchmarks and share them with others to compare the results.

Source code of the Benchmarking Suite is available online [2] under the GPLv3 license.

Benchmark Scripts

A set of benchmark scripts, written in Python, is provided with the Benchmarking Suite. These scripts implement various test scenarios. There are two kinds of tests: performance tests which execute some operation in a loop during a given period of time and efficiency tests for evaluating other aspects Tango. Following performance tests are implemented:

- Attribute Read—counts reads from an attribute,
- Attribute Write—counts writes to an attribute,
- Command—counts command invocations,
- Event—counts event subscriptions,
- Event Push—counts events received at client side,
- Pipe Read—counts read from a pipe,
- Pipe Write—counts writes to a pipe.

The tests listed above can be configured with parameters like test period, number of iterations or number of parallel operations. The actual work is performed by the client processes. The users can switch between clients implemented in Python, C++ and Java. The benchmark scripts can produce reports in CSV and reStructuredText formats.

* Work supported by the Tango Controls Collaboration

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA056

There are two additional tests implemented:

- Attribute Memory—configures multiple dynamic attributes and measures device server's memory consumption (RSS),
- Startup Time—configures multiple devices within a device server and measures server's startup time.

Target Device Servers

Most of the benchmark scripts can test arbitrary devices servers and do not have specific requirements when it comes to the device interface. In order to allow users to compare test results from different Tango installations, a set of standard device servers was implemented. These device servers expose a well-define interface which can be consumed by the client processes of the Benchmarking Suite. The target device servers provide dummy attributes and commands and do not require access to any hardware. Users can choose between device servers implemented in Python, C++ and Java. The target device servers also provide special interfaces required by some benchmarks, like a command for reading memory consumption.

Benchmark Runner

The benchmark runner is a script written in Python that facilitates running the benchmarks. It can read configuration from a YAML or JSON file, start necessary device servers (possibly on remote host, using the Starter device), run one or more benchmarks and then perform necessary cleanup.

Standard Tests

A need for having a set of standard test scenarios was identified. To fulfil this need a set of well-defined tests was prepared. Testing the same scenarios with the same parameters allows users from different institutes to compare the efficiency of their Tango installations. It is expected that such a set of tests will promote collaboration within the Tango Community and contribute to building a directory of performance measurement results which could be used for further analysis.

A set of benchmark runner configuration files defining the standard test scenarios is available online [3].

TANGO CONTROLS PERFORMANCE

The performance of Tango Controls has been measured in a series of tests using the Benchmarking Suite. The test infrastructure and selected test results are presented in this section.

Test Setup

The tests were conducted on Amazon's AWS EC2 Platform [4]. All virtual machines were running on Ubuntu 18.04 (64bit, x86, HVM) operating system. Following software was installed in the virtual machines: Tango 9.3.3, PyTango 9.3.0, JTango 9.5.13, MariaDB 10.04. For each test three separate EC2 instances of various sizes were configured:

- A c5n.2xlarge instance with an SQL database and a DataBase device server,
- an instance with the benchmark runner script and a set of client processes,
- an instance with a device server under test.

The instances were connected into a single VPC network. The test infrastructure is depicted in Fig. 2.

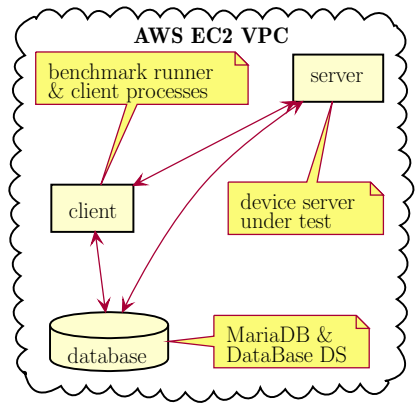

Figure 2: Test setup in AWS EC2.

Instances of type C5n [5] were used in all the tests. C5n instances are optimized for CPU-intensive tasks and offer increased network bandwidth when compared to the classical c5 instances. The hardware parameters of different instance sizes are summarized in Table 1. Regardless of the instance size, the underlying CPU was always an Intel Xeon Platinum 8124M.

Table 1: Parameters of C5n Instances in AWS EC2

Instance	vCPUs	Memory	Network
c5n.large	2	5.25 GiB	25 Gb/s
c5n.xlarge	4	10.50 GiB	25 Gb/s
c5n.2xlarge	8	21.00 GiB	25 Gb/s
c5n.4xlarge	16	42.00 GiB	25 Gb/s
c5n.9xlarge	36	96.00 GiB	50 Gb/s
c5n.18xlarge	72	192.00 GiB	100 Gb/s

The tests focused on measuring attribute read and attribute write speed at both the server and the client side. Each test was configured for 14 iterations. The number of parallel clients increased gradually, from one client in the first iteration up to 128 clients in the last iteration. The duration of each iteration was set to 15 s. Other test parameters were set to default values. In all tests the device server serialization model was set to `BY_DEVICE`.

For server side performance testing, following instances were configured:

- an 18xlarge client instance with client processes implemented in Python,

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA056

- a server instance of size varying from large up to 9xlarge. Each test was repeated with a device server implemented in Python, C++ and Java.

For client side performance testing, following instances were configured:

- an 18xlarge server instance with device server implemented in Python,
- a client instance of size varying from large up to 9xlarge. Each test was repeated with client processes implemented in Python, C++ and Java.

The test results and the scripts used for running the benchmarks on the AWS EC2 platform are available online [3].

Test Results—Server Side Performance

Figure 3 shows the number of read operations performed on device servers running on instances of different sizes. The client was running on an 18xlarge instance. Both client processes and device server were implemented in Python, but a similar trend was observed with device servers implemented in C++ and Java. Figure 4 compares the attribute read performance between Python, C++ and Java server implementations on a 9xlarge server instance.

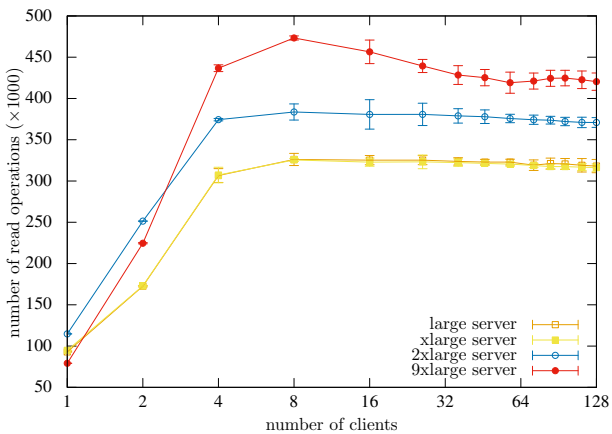

Figure 3: Number of attribute reads during 15 s, Python client (18xlarge), Python server (large-9xlarge).

Figure 5 shows the number of write operations for the same test setup. Also in this case a similar trend was observed with device servers implemented in C++ and Java. Figure 6 compares the attribute write performance between Python, C++ and Java server implementations on a 9xlarge server instance.

With a large number of parallel clients, both attribute read and write speed increase with the increase in the number of CPU cores at the server side. Although 72 CPU cores were available at the client side, running more than 16 client processes does not improve attribute access speed. Device server implemented in C++ offers significantly faster attribute access when compared to Python and Java implementations.

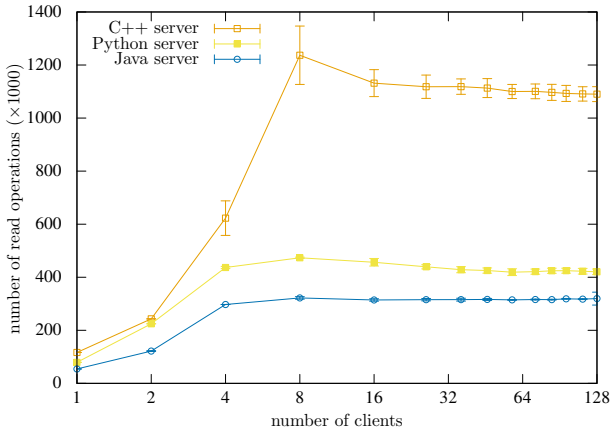

Figure 4: Number of attribute reads during 15 s, 18xl client (Python), 9xl server (Python, C++, Java).

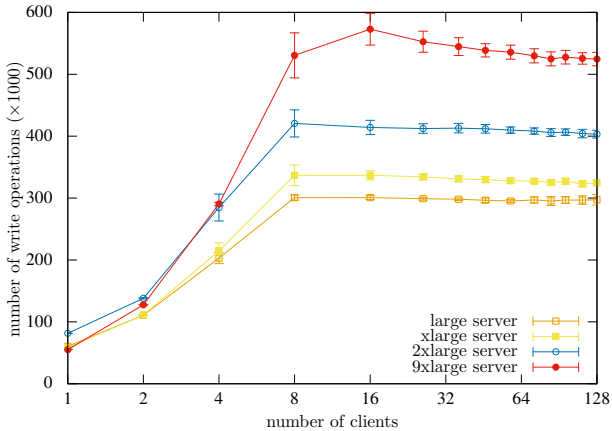

Figure 5: Number of attribute writes during 15 s, Python client (18xlarge), Python server (large-9xlarge).

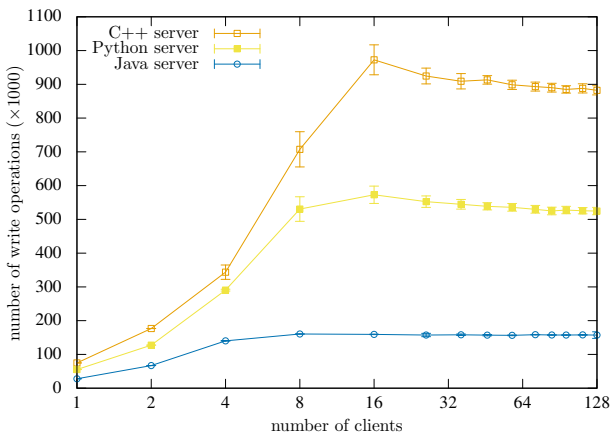

Figure 6: Number of attribute writes during 15 s, 18xl client (Python), 9xl server (Python, C++, Java).

Test Results—Client Side Performance

Figure 7 shows the number of write operations performed by client processes running on instances of different sizes. The server was running on an 18xlarge

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA056

instance. Both client processes and device server were implemented in Python, but a similar trend was observed with client processes implemented in C++. Figure 8 compares the attribute write performance between Python, C++ and Java client implementations on a 9xlarge client instance.

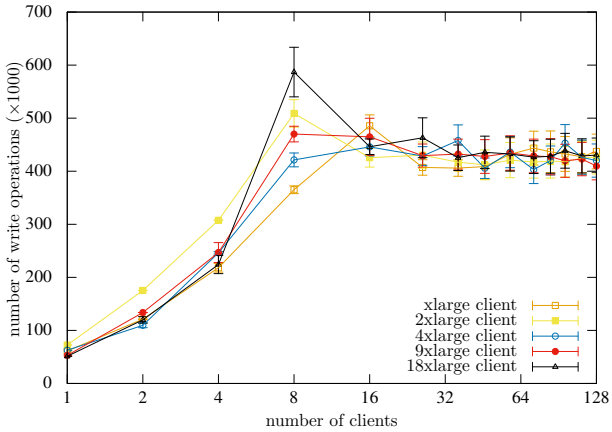

Figure 7: Number of attribute writes during 15 s, Python server (18xlarge), Python client (large-18xl).

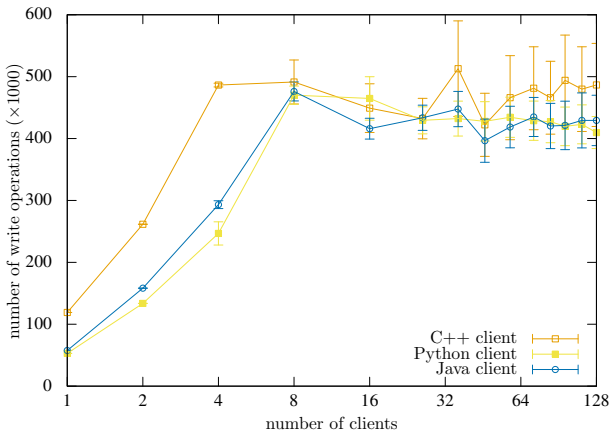

Figure 8: Number of attribute writes during 15 s, 18xl server (Python), 9xl client (Python, C++, Java).

Regardless of the number of parallel clients, attribute write speed is not impacted by the number of CPU available cores at the client side. Even clients running on small instances were limited by the server side performance. Client processes implemented in Python, C++ and Java offer the same attribute write performance.

Similar results were observed for attribute read performance on the client side.

CONCLUSION

The efficiency, performance and resource utilization are important aspects of any control system. The Tango Controls Benchmarking Suite allows for measuring these qualities. The benchmark scripts implement the most common use cases of Tango like reading from or writing to an attribute. The target device servers and the standardized set of test scenarios allow Tango users to share the results and compare efficiency of their systems. The benchmark runner makes it easy to run the benchmark and store test scenarios in simple YAML files. The extensible architecture of the Benchmarking Suite allows to implement new benchmarks easily. More complex test scenarios can be implemented in the future.

The attribute access is an essential operation for most of the devices in Tango and shall be performed as efficiently as possible. A set of tests was conducted to measure attribute access speed. The tests showed that the speed at the server side increases with the number of available CPU cores. In all scenarios, the C++ server implementation is faster than the Python and the Java implementations. Running more than 16 parallel clients does not improve attribute access performance even on multi-core machines. More tests with a larger server instances are needed to evaluate the efficiency of the client side. Also, a detailed analysis of the server side scalability can be performed to determine the speedup on a systems with a large number of CPU cores.

It is important to note that the performance figures are for multiple clients communicating with the same device server. A Tango Control System can start as many device servers as it needs in order to scale up performance. This way almost linear scaling can be achieved i.e. unlimited scaling is possible as long as hardware is available to start more device servers. The only bottleneck is when accessing to the same hardware (or other resource) which does not support multiple accesses in parallel.

ACKNOWLEDGEMENTS

The work presented in this paper would not be possible without the contribution from Jan Kotanski (from DESY, Hamburg) who worked on the first version of the Benchmarking Suite and developed most of the benchmarks. The authors acknowledge Andy Götz (ESRF) who suggested and made the tests on AWS possible and for his comments on the paper.

REFERENCES

[1] TANGO Controls, https://www.tango-controls.org

[2] A Tango Controls Benchmark suite, https://github.com/tango-controls/sys-tango-benchmark

[3] A set of standard tests for Tango benchmark, https://github.com/tango-controls/sys-tango-benchmark-standard-tests

[4] Amazon EC2, https://aws.amazon.com/ec2

[5] Introducing Amazon EC2 C5n Instances Featuring 100 Gbps of Network Bandwidth, https://aws.amazon.com/about-aws/whats-new/2018/11/introducing-amazon-ec2-c5n-instances