Performance Assessment of OMG compliant Data Distribution Middleware

Christian Esposito and Stefano Russo
Dipartimento di Informatica e Sistemistica,
Universita' di Napoli Federico II
Via Claudio 21, 80125 - Napoli, Italy
{christian.esposito, stefano.russo}@unina.it

Dario Di Crescenzo
Consorzio SESM Soluzioni Innovative,
Circum. Est. Napoli, 80014
Giugliano in Campania (NA), Italy
ddicrescenzo@sesm.it

Abstract

Event-Driven Architectures (EDAs) are widely used to make distributed mission critical software systems more-efficient and scalable. In the context of EDAs, Data Distribution Service (DDS) is a recent standard by the Object Management Group that offers a rich support for Quality-of-Service and balances predictable behavior and implementation efficiency. The DDS specification does not outline how messages are delivered, so several architectures are nowadays available. This paper focuses on performance assessment of OMG DDS-compliant middleware technologies. It provides three contributions to the study of evaluating the performance of DDS implementations: 1) describe the challenges to be addressed; 2) propose possible solutions; 3) define a representative workload scenario for evaluating the performance and scalability of DDS platforms. At the end of the paper, a case study of DDS performance assessment, performed with the proposed benchmark, is presented.

1. Introduction

Distributed fault-tolerant real-time software systems are becoming more widespread and increasingly used for a vast range of applications. These systems are characterized by stringent requirements in terms of predictable delivering, timeliness, dependability, data persistency and security. Moreover, they are made of several interconnected elements, so the performance must not degradate when growing the system size. Event Driven Architectures (EDAs) [1] are gaining increasing attraction in these information-driven scenarios, to make systems more-efficient and scalable [2]. In an event-based mode of interaction components communicate by generating and receiving event notifications, where an event is any occurrence of a happening of interest. A significant amount of EDA systems have been developed and implemented, both by industry and by academia [3]. The main weaknesses of these solutions are related to either a limited or not existing support for Quality-of-Service, either to the lack of architectural properties which promote dependability and survivability on a large scale [4]. In order to fill this gap, Object Management Group (OMG) carried out a new standard, namely Data Distribution Service (DDS) [5]. OMG DDS is tasked with ensuring that quality constraints, such as reliability, durability, persistency and so on, are met during event delivery. This is done trying to respect the requirements of a real time communication in terms of predictability. The DDS specification does not outline how messages are delivered, so several architectures are nowadays available. Application developers, who want to determine which DDS product best meet their requirements, needs an effective benchmark for evaluating and comparing competitive DDS implementations. To be useful, a benchmark must fulfill these requirements [6]: 1) Be based on a workload representative of real world applications; 2) Allow reproducibility of obtained results; and 3) Stress all the critical functionality offered by DDS products.

The rest of the paper is organized as follows: 1) an overview of the DDS specification and a comparison of our work with other efforts in DDS benchmarking; 2) a description of the design of the proposed benchmark, namely DDS-Bench, outlining encountered challenges and adopted solutions; 3) a discussion of a representative workload for DDS products; 4) a case study of comparison of two different DDS products; and at least 5) the concluding remarks.

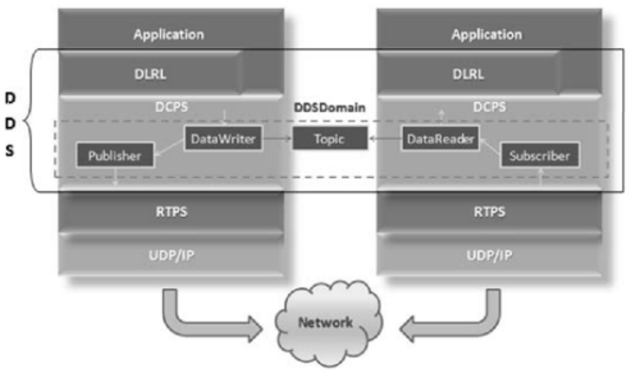
2. Background

2.1 Overview of OMG DDS

Data Distribution Service (**DDS**) [5] is a recent OMG standard for data distribution, adopting the *Publish-Subscribe Pattern* and a *Data-centric View*. It aims to balance predictable behavior and implementation efficiency. As shown in figure 1, the OMG DDS specification describes two layers: 1) *Data-Centric Publish-Subscribe* (**DCPS**)

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

Figure 1: Layers of OMG DDS specification



level, that efficiently delivers the proper information to the proper recipients; 2) an optional *Data-Local Reconstruction Layer* (**DLRL**) level, that is an object-model interface of DCPS functionalities. In particular DLRL allows distributed data to be shared by distributed objects as if the data were local. In the DCPS layer, information flows with the aid of the following entities [7]:

- A *Publisher* disseminates data, received from a sending application.
- A *DataWriter* acts as a typed access to a publisher. It is used by the application to communicate to a publisher the existence and value of data-objects of a given type. The DataWriter - Publisher association expresses the intent of the application to publish the data described by the data-writer in the context provided by the publisher (*Publication*).
- A *Subscriber* receives published data and makes it available to the receiving application.
- A *DataReader* behaves as a typed access attached to the subscriber. The DataReader - Subscriber association expresses the intent of the application to subscribe to the data described by the data-reader in the context provided by the subscriber (*Subsmission*).
- A *Topic* consists of a data type and a name, and connects a DataWriter with a DataReader. Data samples start flowing only when the Topic associated with a DataWriter matches the Topic associated with a DataReader.

DDS applications send and receive data within a *Domain*, a virtual space that connects certain publishing and subscribing applications. Only applications within the same domain can communicate, and this restriction helps isolate and optimize communication within a community that shares common interests. DDS relies on the use of different QoS to tailor the service to the application requirements. In particular QoS parameters allow to tune the robustness of the middleware against the network unavailability and the information timeliness. These DDS QoS policies can be configured at various levels of granularity (i.e., topics, publishers, data writers, subscribers, and data readers), thereby allowing application developers to construct customized contracts based on the specific QoS requirements of individual use cases.

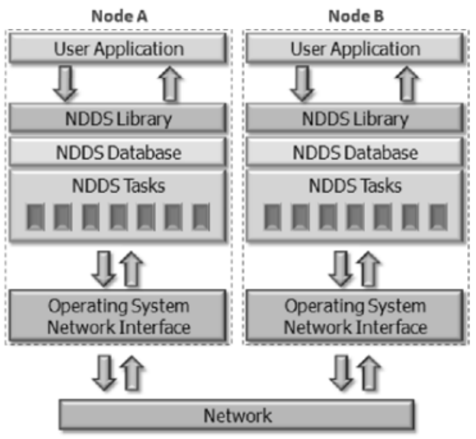
OMG also defined a standard "wire protocol" for DDS-compliant solutions, called *Real-Time Publish Subscribe* (**RTPS**) [8]. It has been realized on a best-effort transport layer, such as UDP/IP, due to delivery timeliness. But the transport is completely pluggable, so it is also possible to adopt a connection-oriented transport layer. In order to enable a reliable communication, even if using a best-effort transport, the protocol exhibits a stateful behavior. Practically, the published message is stored, representing a sort of state, so the protocol is able to retransmit it in the case of message losses. The recovery action in the case of a message loss is performed in an Automatic Repeat-reQuest (ARQ) fashion: if a timeout will be exceed or a NACK will be received, the message is resent.

2.2 DDS Implementations

The DDS/RTPS specification does not outline how messages are exchanged between the communication endpoints. Thus, providers are free to develop their own solutions. In this section we present the three main DDS-complaint platforms, each one characterized by a different architecture.

2.2.1 RTI DDS

Figure 2: Architectural view of RTI DDS

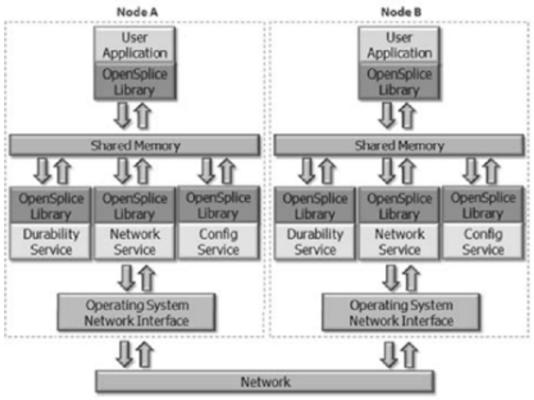


The DDS implementation provided by Real-Time Innovations (RTI) adopts a decentralized architecture. It places the communication- and configuration-related capabilities into the same user process as the application itself. These capabilities execute in separate threads that the DCPS middleware library uses to handle communication and QoS. The middleware is composed of a run-time library, a database, and tasks. The application is directly linked with the library. The tasks perform all of the message addressing, marshalling/demarshalling, and transporting. The database contains all the information needed to delivery a message to the discovered endpoints.

2.2.2 OpenSpliceDDS

The DDS implementation provided by Prismtech uses a federated architecture. It utilizes a shared memory infrastructure to interconnect the applications and a set of services within one computing node. The offered configurable and extensible set of services is composed of: 1) Networking Service: a QoS-driven real-time networking based on multiple reliable multicast channels; 2) Durability Service: a fault-tolerant storage for both real-time state data as well as persistent settings; 3) Config Service: a way to configure the middleware via a XML-File. This architecture decouples the applications from DCPS configuration and communication-related details. The Networking Service works as a daemon, in charge of interconnect the application on a the same node with the outside world. It communicates with DCPS daemons running on other nodes and establishes data channels based on QoS requirements and transport addresses.

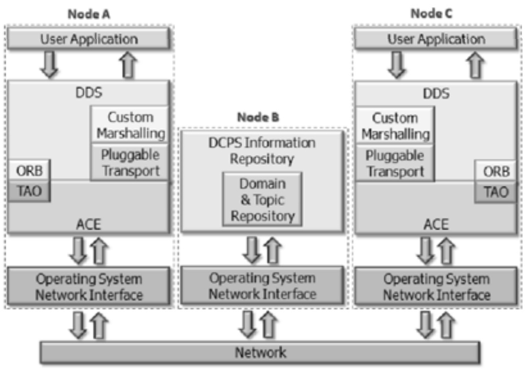
Figure 3: Architectural view of OpenSpliceDDS



2.2.3 OpenDDS

The DDS implementation provided by Object Computing Inc. (OCI) adopts a centralized architecture. It uses a single Information Repository server running on a designated node to store the information needed to create and manage connections between DDS participants in a domain. The data itself passes directly from publishers to subscribers, whereas the control and initialization activities require communication with this daemon server. OpenDDS uses the CORBA interfaces to deliver data.

Figure 4: Architectural view of Open DDS



3. Overview & Design of DDSBench

Choosing the right DDS platform fitting with application requirements is a complex activity, as it is driven by a plethora of criteria such as economic costs, conformance to standard, advanced proprietary features, performance, scalability, etc. To help application designers to select the DDS implementation to use, we propose a benchmark suite, namely **DDSBench**, which objectives are: 1) compare the performance of different DDS implementations; 2) develop a framework that automates all the steps needed to execute a measurement-based performance evaluation of DDS-complaint solutions; and 3) apply workloads that systematically identify performance bottleneck.

The rest of this section is organized into three parts: the first one outlines the challenges encountered during the development of DDSBench and how they have been addressed; the second introduces a representative workload for DDS platforms; while the last one describes the usage of DDSBench to conduct a performance evaluation campaign.

3.1 Challenges and Resolutions

During the design of the DDSBench framework, we encountered several challenges, below there is a list of them, jointly with the adopted solutions:

Workload Configurability There are several usage scenarios for DDS-complaint middleware. This requires that the benchmark framework allows the user to precisely configure and customize the adopted workload according to their requirements.

Solution: DDSBench offers an extensive support to workload characterization in terms of nine parameters, as shown in table 1. The combination of these parameters allows the user to define the proper workload for the desired data-centric use case.

Table 1: DDSBench Parameters

Name	Description
DDS	A DDS solution under benchmark
Message Type	One of the message types defined into DDSBench*
Message Size	Dimension of the size-variable message type
Publication Period	Interval in nanoseconds between two consecutive transmissions
Message Iterations	Number of messages exchanged during a test
Quality-of-Service	Communication Quality during a test**
History	Number of message to store in case of a reliable and transient communication
Ownership	Set ownership exclusive and its strength in case of multiple publishers***
Multicast	Enable a multicast transport

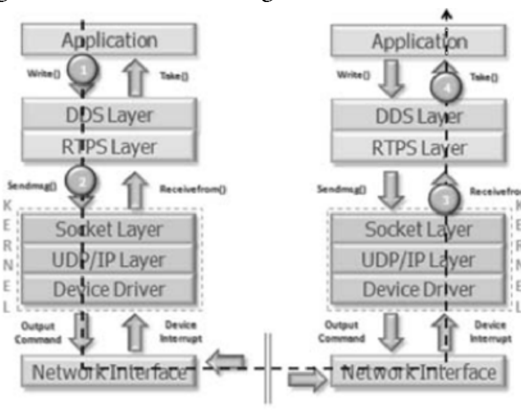
**supported Simple, Complex, CompactAvenue and FullAvenue*
***allowed configuration: Best-Effort, Reliable Volatile and Reliable Transient*
*** *not present in OpenDDS*

Test Parameters Settings Not all the possible combinations of DDSBench parameters are compatible, e.g. if OpenDDS is chosen as DDS type, it is not allowed to choose ownership QoS, because it is not implemented. Moreover, there are incompatibilities also between different DDS QoS [5], e.g. if a publisher uses a reliable delivery, also the connected subscriber have to use it, otherwise the communication will not be established. Manually setting and checking these parameters is tedious and error-prone.
Solution: DDSBench helps in the choice of allowed combinations, indicating the possible presence of incompatibility. Then, all the chosen configurations are stored in an XML file, so they can be used in future.

Execution of Benchmarks A DDS benchmarking test consists of several applications, distributed over the nodes of the testbed. The first problem is to deploy the benchmarking code on all the nodes. The second problem is to launch each application on each node of the testbed. At least, each application generates a log file with the measured latency and throughput values. The last problem is to concentrate in a single point all these files. Without a mechanism for automated execution and collection of benchmark data, these problems are almost impossible to be addressed.
Solution: DDSBench automatically 1) disseminates its code on all the node reported into the testbed configuration file, 2) launches applications via rsh sessions, and 3) transfers via rcp all the generated measurement files into a central repository.

Latency Metric Usually latency is defined in terms of *Round-Trip Time* (RTT), i.e. the time elapsed between sending a message and receiving it back from an echoer. A different metric of latency is the *One-way Transit Time* (OTT) [9], i.e. the difference between the time to send a message on publisher side and the time to receive it on subscriber side. OTT is better than RTT, because it measures the delivery time as perceived by a subscriber. But, it has a serious drawback: the timestamps are taken from two different clock, i.e. one on the publisher machine and the other on the subscriber machine. So this measure is affected by synchronization errors. In the case of DDS Benchmarking, using the *Network Time Protocol* (NTP) [10] is not a solution due to low accuracy.
Solution: The synchronization errors in the OTT estimations are functions of *clock offset*, i.e. the difference between the time taken on publisher side and the one on subscriber side, and *skew*, i.e. the difference in frequency variation between the two clocks [11]. The solution is to remove this two factors from the obtained measures with a Skew&Offset Removal Tool. The skew is estimated and removed using the Moon algorithm [12], while the offset is computed with the procedure outlined in [13].

Figure 5: Route of a message to reach a subscriber



Deeper Latency Analysis When a message is delivered to a subscriber, it follows a precise route, as illustrated in figure 5. After a message has been passed to a DataWriter, it passes through the DDS and RTPS layers, where several operations, such as marshaling, segmentation and others, are performed. Then, the message enters the kernel space, and through the Network Interface is sent on the network. The same way is done in the opposite direction to reach the target DataReader. The OTT measure summarizes with a number all these contributions. However, a deeper analysis is needed in order to assess how the architectural choice of a specific DDS solution can influence the overall performance.
Solution: OTT can be divided into three contribu-

tions: 1) Sending Overhead (SO), i.e. time elapsed between the invocation of write() on the DataWriter and the sendmsg() system call; 2) Transmission Overhead (TO), i.e. time elapsed from the sending of a message in the socket at publisher side to its receiving at receiver side; and 3) Receiving Overhead (RO), time elapsed between the returning of receivefrom() system call and the invocation of take() on the DataReader. Considering the timestamps in figure 5, the performance is characterized as:

$$OTT = SO + TO + RO$$
$$\begin{cases} OTT = T_4 - T_1 & SO = T_2 - T_1 \\ TO = T_3 - T_1 & RO = T_4 - T_1 \end{cases}$$

For this detailed performance analysis, It is needed obtaining the invocation and returning time of a system call. This is possible using SystemTAP[1] [14], i.e. a free software (GPL) infrastructure for tracing and probing system calls on Linux systems. It provides a way to name events, and to give them handlers. Whenever a specified event occurs, the Linux kernel runs the handler as if it were a quick sub-routine, then resumes. It ensures a low invasiveness on the system, and adds a negligible delay to the normal execution time of system operations. DDSBench provides a set of a probes realized as an handler, that print the raising time of the associated event.

Outliers Presence The obtained measures could be affected by *outliers*, i.e. values out of range respected the others measures. This can distort the statistics on the measurement data set.
Solution: The solution is to use robust statistic [15], i.e. representative of the real trend into the data set, and not function of the outliers. For this reason, to analyze the central tendency of a data set the median has been chosen, and for the dispersion the InterQuartile Range (IQR).

Implementation Heterogeneity The DDS implementation differs in terms of adopted architectures, so there are also differences in their use, e.g. OpenDDS needs the activation of an Information Repository and the dissemination of its IOR reference to all the applications. This is an important matter, because if the implementation is not well used, the communication will not be established, e.g. if a daemon is not launched on all the testbed nodes, some applications could not communicate with the others. Some of these troubles are critical, because the user could not be aware of an error in the implementation usage. So the benchmark have to shield the user from the implementation heterogeneity.
Solution: DDSBench has a script to launch a test given the chosen parameters. It will do all the needed actions to establish a successful communication depending on the chosen DDS implementation, masking product-specific operations.

[1]http://sourceware.org/systemtap/

3.2 Workload

The major goal of DDSBench is to provide a standard workload to evaluate the performance and scalability of DDS-complaint middleware. The workload to consider has to satisfy two requirements [16]: 1) *Representativeness*, i.e. it reflects the way DDS implementation are used in real-life systems; and 2) *Comprehensiveness*, i.e. it exercises all the functionalities typically used in the major classes of information driven applications. The benchmarking scenario chosen for DDSBench models the interactions within an *Air Traffic Management* (ATM) system. It offers an appropriate means to stress different subsets of the functionality offered by DDS entities.

An ATM system is characterized by means of several computing elements in charge of different ATM tasks. These elements are mutually dependent, i.e. to perform its own tasks, they need information owned by others. A recent project supported by EuroControl, called *AVENUE*[2], has standardized the interactions in an ATM system with two kind of messages. We called the first one FullAvenue, that is characterized by a size of almost 78KB and an extremely complex structure; while the second one is called CompactAvenue, and is a smaller version of the first one with a size of almost 28 bytes. Considering the high number of tasks that this system have to perform, in order to monitor flights in a region of airspace, the messages are exchanged very frequently, e.g. the publishing rate is almost 100 messages per second. Due to the mission critical nature of the system, these continuous interactions are subject to strong availability requirement: the offered QoS have to be the highest possible, i.e. reliable delivery with a durability of all the sent messages. An interaction is defined in terms of only one publisher and at least six subscribers. There is only one interaction participant per node, in order to guarantee diversity and avoid simultaneous failures.

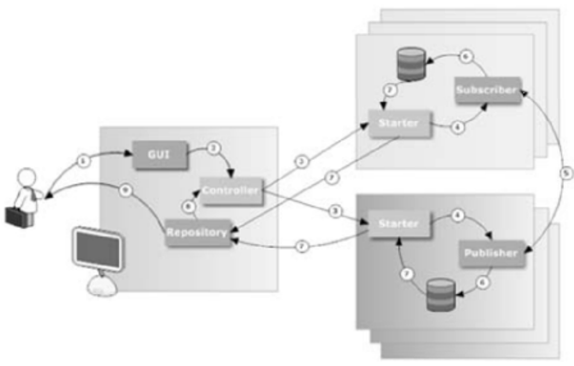
3.3 DDSBench usage

To perform a performance assessment with DDSBench, there are several steps to pass through, as shown in figure 6:

1. The user has to write an XML file with the values of all DDSBench parameters. A test session is composed

[2]http://www.eurocontrol.int/eec/public/standard_page/ERS_avenue.html

Figure 6: Usage of DDSBench



by several tests, each one represented by a row in the XML file. A Java-based GUI aids the user in this task, looking on the fly for eventual incompatibilities. Then the user can define the nodes that will compose the testbed. The tool automatically checks if each node is reachable and/or the DDS implementation is properly installed.

2. The GUI activate the Controller, that manages all the test campaign. The GUI passes to the Controller the XML configuration file, testbed description file and the archive with the benchmark code.

3. The Controller distributes the code and activate a Starter on each node of the testbed.

4. First the Starter uses NTP to synchronize the node to a NTP server. It is done to minimize the clock offset. Then It launches a DDS application, based on a test description given by the Controller. This description contains the number and type of applications to launch and the parameters to pass them.

5. After concluding the startup, the DDS applications exchange messages and measure the performance metrics.

6. When the transmission is terminated, each DDS application stores a report file with all the measures into a local repository. The Starter uses the Skew&Offset Removal Tool to derivate the OTT measures from the RTT ones.

7. The Starter collects all the reports in its local repository and sends them to the central repository.

8. If there are more tests to perform, the control is handed down the Controller, and the step 3 is repeated using an other row of the XML configuration file.

9. If the test session is concluded, the Repository processes the collected measures in order to generated automatically charts and reports, summarizing hardly-finished campaign.

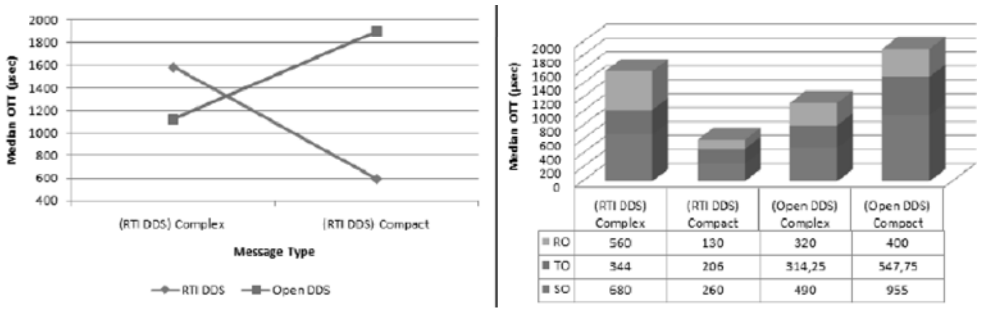
4 Empirical Results

In this section we present three experiments made using DDSBench to evaluate the performance of two DDS products: RTI DDS and Open DDS. The experiments presented in this section were conducted using workstation with 2 Xeon 2.8 GHz (with Hyperthreading), 6 Gb of RAM and Red Hat Enterprise Linux 4. The machines are interconnected with a Gigabit Ethernet. For the results to be valid, the publisher and subscriber machines are identical with respect to hardware and software setup.
During each test, 500 messages are exchanged and the first 10% of messages are discarded in order to avoid measures distorted by cold start issues or warm-up effects [17]. At each iteration, the publisher sends a message to the subscriber, storing the sending timestamp. Then the subscriber receives it, stores the receiving timestamp, makes a copy of the received message, store a sending back timestamp and sends back the message to the publisher. At last the publisher receive back the message and store the receiving back timestamp. These timestamps and the Skew&Offset Removal Tool allow to evaluate the message OTT.

4.1 Marshaling Tests

The scope of this experiment is to study the influence of the message structure on the achievable performance. The experiment is performed according to the previously defined workload, except that the publisher sends a CompactAvenue message, while in a second test it sends an octet stream (Complex) with the size of a CompactAvenue. The CompactAvenue, indeed, is a good workload to stress the marshaling function. As illustrated in the left side of figure 7, the middleware under benchmarking show different behaviors: Open DDS has a performance worsening, while RTI DDS has an opposite trend. It is impossible to understand the reason of this on the base of the only OTT value. If the OTT is decomposed in three factors, as described in section 3.1, we can see a decreasing of TO in the case of RTI DDS, when there is an increasing for Open DDS. This is due to a less number of bytes sent over the network. Also other factors have the same trend. So RTI DDS adopts an optimized marshaling: even if we have the same message size, in the case of CompactAvenue the marshaling phase is faster, because a part of the message is discarded. The Avenue message is defined in terms of switch statements, and even if its nominal size is 27812, its real one is lower. RTI DDS takes advantage of this singularity, and discards

Figure 7: Test to study the influence of message structure on the performance



	(RTI DDS) Complex	(RTI DDS) Compact	(Open DDS) Complex	(Open DDS) Compact
▨ RO	560	130	320	400
■ TO	344	206	314,25	547,75
■ SO	680	260	490	955

the parts of the message without information. In the case of Open DDS, all the message is considered, but the marshaling takes more time, due to the structure complexity.

4.2 Message Size Tests

The scope of this test is to assess the performance of each implementation varying the message size. The configuration is always the same, the exchanged message is an octet stream with a variable size. As presented in figure 8A, incrementing the message size leads to an exponential increase of median OTT, lightly heightened in the case of RTI DDS. But jitter tells us that RTI have a more stable behavior than Open DDS, even if for the last message we have an high increasing for RTI.

4.3 Scalability Tests

The scope of this test is to evaluate the changes in performance when the number of nodes grows. An octet stream of 8192 bytes is exchanged under the same configuration, but in this test there is more than one subscriber, and each one is placed on a distinct node of the testbed. As shown in figure 8B, the median OTT has a floating trend almost similar in both the solutions. The jitter has an interesting behavior: after an initial increasing, is almost constant for RTI DDS, while Open DDS experiences an almost linear increase with the growing of the nodes numbers.

5 Related Work

The scientific literature lacks papers on DDS benchmarking. Two technical projects are worth mentioning, that have proposed relatively complete benchmarking frameworks. Neither of them however addresses all the challenges outlined in section 3.1

Open Architecture Benchmark. Open Architecture Benchmark (OAB) [18] is a DDS benchmark effort associated with the Open Architecture Computing Environment, an open architecture developed by the US Navy. The objectives of this work are to understand ability of DDS-complaint products to support bounded latencies and sustained throughput required by combat system applications and to provide a baseline of comparison between different DDS solutions. The analyzed DDS solutions are RTI DDS and OpenSplice DDS.

DDS Benchmark Environment. DDS Benchmark Environment (DBE) [19] extends OAB effort by (1) including a third DDS implementation, the one provided by OCI, in the comparisons and (2) classifying the different architectures used by these implementations. Such as DBE, our work uses all the three main DDS implementations, but differs in terms of the following characteristics: 1) the performance is evaluated by means of One-way Transit Time (OTT) instead of Round-Trip Time (RTT); 2) measures are elaborated automatically off-line; 3) a Java-based application helps the user to define the testbed and choose the test parameters; 4) performance is assessed for large and complex data.

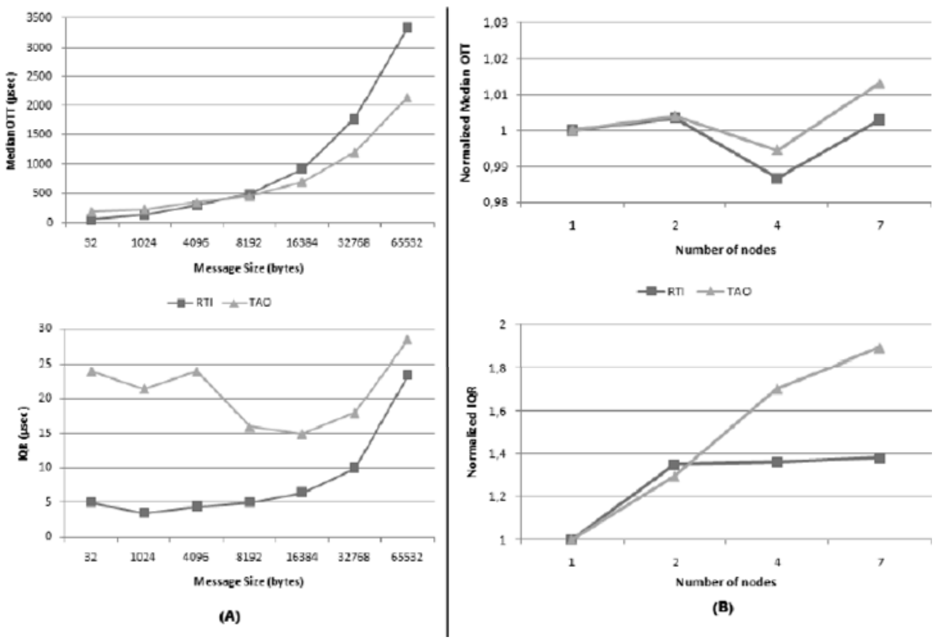
6. Concluding Remarks

The paper focused on performance assessment of OMG DDS-complaint data distribution middleware. The main contribution of this paper was to present an expertise report on the design and implementation of benchmark for DDS middleware. This task is characterized by several problems to be addressed, and a set of possible solutions has been proposed. The experiments performed to compare two DDS implementations demonstrated that a federated architecture can lead to more predictable performance. Future work is to extend DDSBench to evaluate other performance properties, such as throughput and determinism, significant to a DDS-complaint middleware.

7. Acknowledgments

This work has been supported by the University of Naples and by FINMECCANICA in the framework of the

Figure 8: Test (A) varying the message size and (B) increasing the number of nodes



project "IniziativaSoftware"[3]. The authors would like to thank Antonio Strano for his excellent contribution in developing the Skew&Offset Removal Tool.

References

[1] G. Muhl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer, 2006.

[2] Hype cycle for application development. *Gartner - Research Report*, 2007.

[3] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. *The Computer Journal*, 28(5), 602-626.

[4] S.P. Mahambre, M. Kumar, and U. Bellur. A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware. *IEEE Internet Computing*, 11(4):35–44, July-August 2007.

[5] Data Distribution Service for Real-time Systems, version 1.2. *Object Management Group (OMG) - Technical Document*, August 2007.

[6] S. Knouvev. *Performance Engineering of Distributed Component-based Systems - Benchmarking, Modeling and Performance Prediction*. Shaker Verlag.

[7] G. Castellote. OMG Data-Distribution Service: Architectural Overview. *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, pages 19–22, May 2003.

[8] The Real-Time Publish-Subscribe Wire Protocol, DDS Interoperability Wire Protocol, version 1.0. *Object Management Group (OMG) - Technical Document*, Mars 2006.

[9] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way delay metric for ippm. *RFC 2679, Internet Engineering Task Force (IETF)*.

[10] D.L. Mills. On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system. *ACM Computer Communication Review*, 20(1):65–75, 1990.

[11] V. Paxson. Measurements and analysis of end-to-end internet dynamics. *Ph.D. thesis, University of California, Berkeley*, 1997.

[12] S.B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. *Proceedings of 16th IEEE Conference on Computer Communications (InfoCom 99)*, 1999.

[13] M. Tsuru, T. Takine, and Y. Oie. Estimation of clock offset from one-way delay measurement on asymmetric paths. *Proceedings of the Symposium on Applications and the Internet Workshops (SAINT 02)*, pages 126–134, 2002.

[14] W. Cohen. Instrumenting the Linux Kernel with SystemTap. *Red Hat Magazine*, September 2005.

[15] D.L. Massart, J. Smeyers-Verbeke, X. Capron, and K. Schlesier. Visual Presentation of Data by Means of Box Plots. *LC-GC Europe*, 18(4):2–5, 2005.

[16] K. Sachs, S. Kounev, M. Carter, and A. Buchmann. Designing a Workload Scenario for Benchmarking Message-Oriented Middleware. *Proceedings of the 2007 SPEC Benchmark Workshop*, January 2007.

[17] A. Buble, L. Bulej, and P. Tuma. CORBA Benchmarking: a course with hidden obstacles. *Proceedings of the IPDPS Workshop on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (PMEOPDS 03)*, pages 279–284, April 2003.

[18] B. McCormick and L. Madden. Open Architecture Publish-Subscribe Benchmarking. *OMG Real-Time Embedded System Workshop (RTES 05)*, 2005.

[19] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. Schmidt. Evaluating technologies for tactical information management in net-centric systems. *Proceedings of SPIE: Defense Transformation and Net-Centric Systems, Editor Raja Suresh*, 6578:657–668, May 2007.

[3]http://www.iniziativasoftware.it/