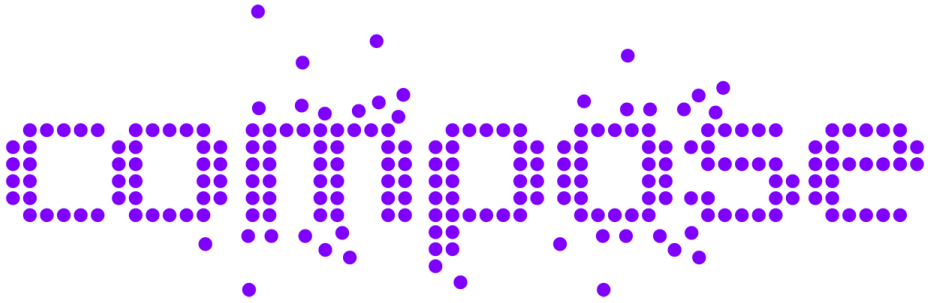


Collaborative Open Market to Place Objects at your Service

D4.2.3

Enhanced prototype of the COMPOSE communication infrastructure

Project Acronym	COMPOSE	
Project Title	Collaborative Open Market to Place Objects at your Service	
Project Number	317862	
Work Package	WP4 Infrastructure Development and Prototyping	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewer	Luca Panziera	OU
Reviewer	Wolfram Gottschlich	UNI PASSAU
Dissemination Level	PU	
Contractual Delivery Date	30/04/2015	
Actual Delivery Date	30/04/2015	
Version	V1.0	

Abstract

The communication infrastructure provides membership and communication patterns of various kinds to the different entities running within the COMPOSE platform. This document aims to accompany the enhanced prototype of the communication system rather than provide a detailed design document of the demonstrated system. A full-fledged detailed design document is scheduled for the end of the project.

The developed communication infrastructure is based on the creation of a peer-to-peer overlay network, in which each communication node connects and guards a small amount of peer nodes, and regularly exchanges information with them. That construct identifies and disseminates information as to nodes joining or leaving the overlay. In addition, the specific interest of different nodes is exchanged by gossip with other nodes in the overlay, forming an efficient interest aware membership service. In addition, a structured topology is maintained within the overlay which enables efficient key-based routing, which most notably is used for an efficient implementation of the pub / sub communication paradigm.

The communication infrastructure is mainly used within COMPOSE to connect between the service objects layer and COMPOSE applications. Compose applications can declare their interest in obtaining change event notifications on service objects of their interest, and the entire exchange of timely information will be performed via the communication infrastructure. In addition this service is used as a back-bone for the monitoring capability infrastructure.

The full cloud deployment consists of a coordinator which serves as a bootstrap for this service. In addition communication servers are deployed as cloud applications, and interested entities can connect to the communications infrastructure and thus participate in the information exchange by using purposely built communication clients, which are available both in Java and JavaScript.

Document History

Version	Date	Comments
V0.1	15/03/2015	Initial version
V0.2	01/04/2015	Added tests
V0.3	15/04/2015	Incorporated comments from Passau
V0.4	28/04/2015	Incorporated comments from OU
V1.0	30/04/2015	Incorporate all comments

Table of Contents

List of Figures

Acronyms

Table 1: Acronyms table

Acronym	Meaning
COMPOSE	Collaborative Open Market to Place Objects at your Service
DHT	Distributed Hash Table
JVM	Java Virtual Machine
NACK	Negative Acknowledgment
P2P	Peer-to-Peer
Pub/Sub	Publish/Subscribe

1 Introduction

The communication infrastructure is meant for the scalable dissemination of membership information as well as several communication paradigms to the different users encompassing the COMPOSE platform. This service is available as an additional cloud service available to all interested applications and components, such as the platform administration itself for management purposes, and different COMPOSE entities embedded within the platform for communication and for achieving self-management properties. We opted for a peer-to-peer (P2P) based overlay as the underlying structure of the communication system. An overlay network is made of a group of processes that connect to each other, whereby each process connects only to a small number of its peers. This architecture was mainly chosen for its contribution to the overall scalability of the system.

Services provided by this component at this time are membership and publish / subscribe (pub / sub). Later releases of this component may add (i) point-to-point messaging either over the overlay links or directly between the peers by opening a specific connection between the source and the target, (ii) broadcast, for enabling the submission of a message to all participating entities; a capability mainly used for management purposes. A central design point is to have a unified technology that will cover the entire spectrum of communication capabilities required for the proper and efficient running of the platform. That includes pure communication capabilities, as well as membership and monitoring components.

The design followed is of a scalable, fully distributed, messaging, membership and monitoring infrastructure. This scalable and distributed infrastructure utilizes peer-to-peer and overlay networking technologies to perform its operations at the scale and performance envisioned, as can be seen in Figure 1. In traditional small scale systems, typically a full connections mesh is used in which each communication node is connected to each other node in the system. These are typically centralized solutions in which either IP multicast is used or in contrast multicast by unicast methods are used. Such a scheme is not feasible anymore when moving to a large scale, in which case we opt for a self-organizing, structured overlay network providing overlay-based communication services. In addition, strong consistency semantics which are feasible in small scales become a hinder to scalability, thus in this large scale we opt for an eventual consistency semantics for the membership information. Individual group communication sub-systems can implement a variety of messaging semantics, which in turn can be used by different clients based on the needs of their own system. In the current pub / sub implementation both bets effort and an ACK based semantics are supported (by which an acknowledgment is sent by each node receiving a message).

The system demonstrated at this time consists of a single overlay group, which based on the underlying architecture can scale efficiently up to a couple of thousand members. In addition a hierarchy is built on top of the group concept presented at this time to achieve higher scalability. The current aspect of the hierarchy consists of thin clients connecting to a server which belongs to the group. That will enable thin clients to participate in the global communication scheme without incurring a large overhead. Thus, currently the communication sub-system is installed on multiple servers running inside the platform, and smaller COMPOSE entities communicate with this communication servers based on their needs. This setup supports adequately the system requirements out of the communications component.

The main services envisioned by this component are membership, scalable group communication, and monitoring back-bone dissemination infrastructure. This set of services enable:

- The operation and orchestration of resiliency aware applications, providing fault tolerance – may be used by the platform controller component
- Monitoring, load balancing, resource management, and efficient components scheduling - may be used by the platform controller component
- Application integration and cooperation, by using the pub / sub service.

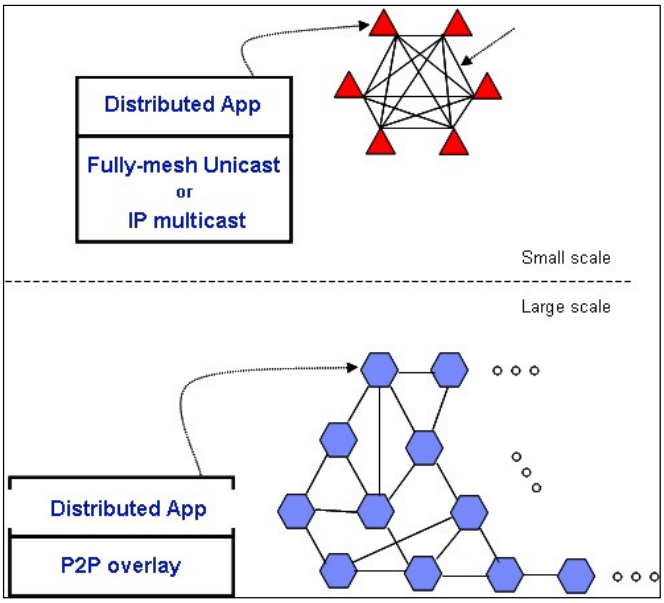


Figure 1: Communications Infrastructure

The current prototype demonstrates a tier of communication servers forming a single overlay group. In addition a thin-clients tier is connected to the servers tier, via a coordinator node. All participating entities collaborate to create a structured topology and exchange information in an eventually consistent manner.

In this prototype the server nodes as well as the coordinator are written purely in Java, so that a proper JVM is needed wherever this is to be deployed. In addition client libraries are implemented and demonstrated both in Java and in JavaScript. In addition, as the prototype is of a networking nature, it relies on the existence of the TCP stack and the availability of free ports, one per each participating communication node. Moreover, at the moment the assumption is that all nodes are in the same administrative domain and do not have to cross firewalls.

Communication between the nodes is performed via TCP, and the information is transmitted as is. Thus, if stronger security requirements are in place, higher layers need to take care of this. For example, encryption and decryption of messages can take place at the applications using this infrastructure for group communication.

The main purpose of this document is to accompany the technical demonstration and provide the necessary background information to be able to put into the right perspective within the COMPOSE platform. It is not intended to be a full-fledged design document of this component. Some more details can be found in the final architecture deliverable (D1.2.2)[1], and a full-fledged design document is expected at the end of the project.

[1] http://www.compose-project.eu/sites/default/files/publications/D1.2.2%20-%20Final%20COMPOSE%20architecture_0.pdf

1.1 Criteria for selection of technologies, Important features and Fit with the project requirements

The first criteria kept in mind for the selection of the proper communications technology was scalability. We wanted to be able to support a large amount of participants, connecting and disconnecting from the communications network, while sending periodic updates which are of interest to a small sub-set of other nodes. For these reasons we opted not to rely on a full mesh topology, but rather an overlay in which every node is connected only to a sub-set of its peers and exchanges periodic information only with its neighbours. In addition we did not want to rely on the existence of IP multicast for information distribution since that capability is switched off in many realistic settings, and in addition does not support potentially wide area distribution. In essence the design goal was for a communications system which exhibits no centralized point, and does exhibit many self-* capabilities.

The chosen solution developed exhibits the desired qualities mentioned above. For operational aspects a joining node needs only to know the information of one node which is currently alive and belongs to an overlay, for the new node to be able to join and automatically find its place within the overlay. Thus, the solution is self-organizing. All nodes in the overlay know how to adjust their connectivity in response to nodes joining or leaving the overlay.

The scalability requirement led also to the choice of an eventually consistent semantics. Keeping strict consistency semantics is often easier for higher layers to grasp and work with, but it usually contradicts a requirement for supporting scalability in an efficient manner. Thus, we decided to loosen up the consistency semantics in a trade-off to gain on the scalability wagon.

In addition a cloud friendly system is of essence, such that both deployment aspects within the cloud can be handled gracefully and the use of it by applications and additional entities within the cloud is made straight forward.

The main group communication flavour we were looking for is pub / sub. The main use for this component is to connect between the service objects and the applications layers in a distributed loosely coupled manner. We wanted a complete disassociation between the different layers, and without relying on different entities knowing each other or being able to communicate with each other. Thus, the ideal communications component will enable the rest of the COMPOSE platform entities to send information without having to know who is interested in that information and how we can ensure that the information gets to the target, and on the other hand we want to be able to subscribe to a piece of infraction without knowing the producer of that information and without necessarily understanding how can we communicate with that entity.

The desire to have scalable pub / sub capabilities led us to choose a structured overlay network. This kind of networks takes slightly longer to construct after nodes joining or leaving but they exhibit superior performance and latency characteristics for information dissemination within the system once it has stabilized.

As mentioned above a central design point was to have a unified technology that will cover the entire spectrum of communication capabilities required for the proper and efficient running of the platform. That includes pure communication capabilities, as well as the membership and monitoring components. Thus, the entire solution was designed and built to fit the platform requirements.

2 Prototype overview - short description

The prototype is centred on demonstrating the membership and pub / sub capabilities of the communication overlay group.

- The prototype will demonstrate the manner in which nodes join and leave the group in an orderly manner.
- Nodes crashing and their effect on the group as well as the dissemination of the relevant information will be demonstrated.
- Demonstrating these capabilities both using the communications server nodes as well as the clients.
- Pub / sub capabilities will be demonstrated.
- Deployment and integration within the COMPOSE platform will be demonstrated. Subscriptions and publishing across COMPOSE layers will be demonstrated.
- Integration with the developers' portal will be exhibited as well.

2.1 Architectural responsibilities and interactions

This component will reside at the lowest layers of the COMPOSE platform run-time and will serve as a generic service to be used by other COMPOSE components, as is illustrated in Figure 2. The cloud in which the COMPOSE platform runs contains its own messaging infrastructure, which is used for the proper functioning of the platform itself. Complementary, the richer functionality we provide will be used by COMPOSE entities such as applications and Service Objects.

First and foremost, the platform run-time management will make use of this component to maintain information on entities which are alive and those that have left or crashed (will further be elaborated in the monitoring deliverable, "D3.2.2.2 - Prototype of the service monitoring tools"). The main interaction of the monitoring aspect of this component is with the platform run-time management system that can figure out which components are alive. Other entities can register as well to obtain that information. The main interaction of the pub / sub capability is between the service objects layer within the data management component and COMPOSE applications running inside the COMPOSE platform cloud infrastructure.

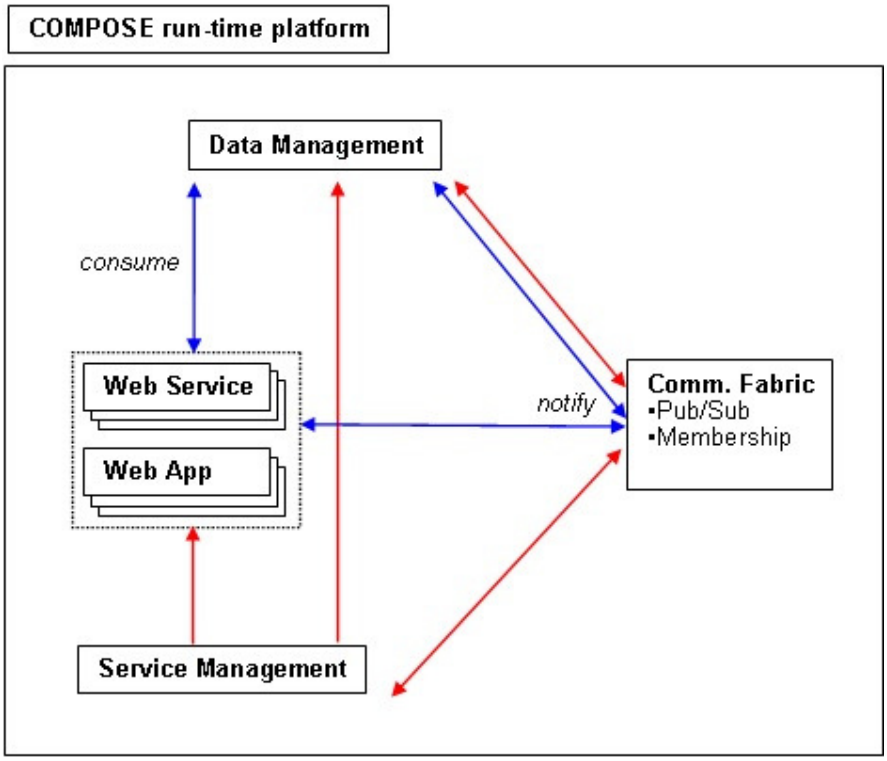


Figure 2: Internal platform interactions

Pub / sub communication can be used by platform entities as well. It can serve as a channel through which de-coupled services or applications get to cooperate. Using such a capability enables the communication and coordination between distinct entities which do not have to be aware of each other, but only need to agree upon the topic (or channel) through which they will be communicating, and the data format they will be using. On the other hand it can serve as a notification mechanism to enable registered entities to find out when certain events have finished and potentially obtain the results of the preceding operation.

2.2 High level design

The design is of a fully distributed, self-organizing, overlay network that does not rely on IP multicast. The group members are organized in a ring based on their virtual ID and each node is connected to its successor and its predecessor. Whenever a node fails or a new node joins the overlay the connections to the successor and predecessor are updated or established if necessary. That ensures connectivity and reachability within the virtual ring. Furthermore, in order to enable efficient key-based routing and information dissemination within the group a structured overlay topology is created and maintained, in which each node connects to a logarithmic number of its peers. These connections enable the efficient dissemination of data, since a correct construction of such a topology will enable each hop in the message route to at least half the distance to the ultimate message target. Several separate topologies can be supported and co-exist, such as adding a random topology layer which may handle the gossip traffic which is used for information dissemination purposes, such as membership and attributes replication. A random topology does not exist within the current prototype.

In order for a new node to join the group it has to know only the address of a single group node which is alive. Once the node is connected to the group, it can determine its location within the ring and all the peer connections are created. Thus, each node is guarded by a small number of peer nodes. Upon a node leaving or failing, the information will be captured by its peers and transmitted to the rest of the group members via a gossip mechanism. Additional options for fast failure detection exist, such as use message flooding for that information or designate a single node to collect that information and have the information transmitted to it directly in addition to the gossip mechanism.

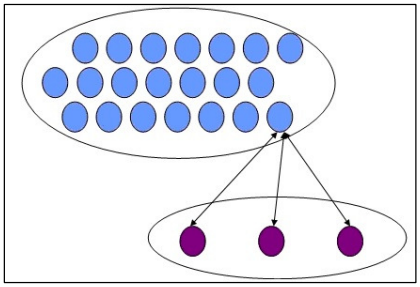


Figure 3: Thin client-server hierarchy

The external API, discussed in section 5, enables the registration of listeners to obtain the information on nodes leaving or joining the group. In addition, the API enables the creation of pub / sub topics, publishers and subscribers. The underlying infrastructure ensures that published messages will be delivered to all interested subscribers in an efficient manner.

The design called for the introduction of another type of entity, namely thin clients. The main reason for the introduction of a thin client notion in addition to the existing communication server is that many of the entities running in COMPOSE that need to be connected to the network are light weight and having them bear the full-fledged communication stack would be excessive. Thus, the servers will form a group of their own in which each member is connected only to a limited amount of its peers. The servers group will have full membership knowledge, providing eventually consistent semantics. The introduction of thin clients enhances the scalability of the system while enabling smaller COMPOSE entities to participate in the communication scheme without bearing the complete burden of the full-fledged communication module. The communications client API enables these clients to participate in the group communication and monitoring activities as participants, able to obtain and broadcast information. This design aims to balance the desire on the one hand to have different entities participate in the communication bus, while keeping in mind that some of the participating entities are small and thus cannot be required to embed within them a full-fledged communication server. Monitoring of all the different kinds of components in the platform is made possible and the servers' group maintains a complete view of the state of the system and all its components.

This scheme, by design, scales up to a couple of thousands in the servers tier. In higher numbers delays in the information propagation may appear as the amount of content that needs to be transmitted by gossip throughout the group becomes rather large. Each server can support numerous smaller scale devices, thus reaching the envisioned scale of simultaneous interactions within the platform.

2.2.1 Main functionalities introduced

Membership

The main goal of the Membership service is to supply members of the group with the information regarding connected processes. This is needed for creating the view of which processes are currently alive, and obtain some information they want to share about themselves. Such information may aid higher layers in several aspects of the management of the system, such as supporting fault tolerance. In order for the system to scale better we opted for an eventual consistency semantic in which after a certain amount of time has elapsed with no

processes joining or leaving the system all nodes will have the same membership information. In addition, to support pub / sub, the interests of the different nodes is maintained and propagated by the same membership gossip mechanism, forming an interest aware membership service. The interests of a node include the topics to which it wishes to publish as well as the topics from which it wants to receive messages.

A joining node needs to communicate with at least one member node which is alive. Once such a connection has been established, both nodes exchange information and based upon that the joining node starts establishing connections to additional nodes in the group, and may cause changes in the connectivity of existing nodes.

Every node in the group maintains open connections to only a small amount of its peers and guards them via a configurable heartbeats mechanism. Once a node ceases to receive heartbeats for a configurable amount of time it declares the peer to be suspect. Information concerning nodes that have joined, left, or crashed is transmitted periodically by each member to its connected peers, and thus the information is spread via a gossip mechanism throughout the cluster in a logarithmic time scale.

Such a capability is important for the platform administration to get a view of which entities are alive, and which others have disappeared. This information has implications for example on an orchestration engine, a recommendation and a composition engine.

Group communication

Efficient broadcast and publish / subscribe can be based on key-based routing. Once a structured topology has been established, efficient routing can be performed based on that topology.

Efficient pub / sub routing is achieved using information maintained by the interest aware membership service. Such a service was implemented and is demonstrated via the pub / sub capabilities. The interest aware membership calls for not only notifications on nodes joining or leaving the system, but also on nodes changing their interest, for example becoming a publisher or a subscriber of a specific topic. Such a service can be used to coordinate between different entities interested in the same information distribution, without knowing all entities in advance or have direct connections between them.

Monitoring

The monitoring back-bone is supported by this communication component. The external API is agnostic to the internal mechanism used to disseminate monitoring information, such that upon need these internal mechanisms can be changed without having external effects. At the current implementation the internal mechanism used for disseminating monitoring information is the pub / sub capability. Producers of information declare themselves as publishers, while consumers of information declare themselves as subscribers. Both edge entities need not be aware of each other but rather need to agree only on the channel via which the information will be exchanged (topic), and the data format to be used such that the semantics can be similarly deciphered by both the producers and consumers of monitoring information.

A more complete design and prototype description is being developed in conjunction with the monitoring task taking place in WP3.2, and will be detailed in a respective deliverable (D3.2.2.2 - Prototype of the service monitoring tools).

3 What is being demonstrated

In this prototype of the communication technology, we concentrate on demonstrating the component's capabilities and their integration within the rest of the COMPOSE platform. This prototype comes to summarize a thorough design and rigorous testing that were performed for these capabilities to ensure that the platform can rely on the proper functioning of these capabilities. Specific aspects of scale and performance were taken into consideration during the design process, and some relevant numbers will be shown.

The tests were designed to verify major functionalities of important classes, each layer separately, and a combination of layers working together. Examples of internal layers include (i) a communication layer, in charge of the low level network communication, (ii) topology layer which creates and maintains the desired topologies, (iii) gossip layer which is in charge of information dissemination and reconciliation (iv) pub / sub layer on top of the topology layer.

The main capability that is being demonstrated is pub / sub, along with the integration within the COMPOSE platform. The two main integration points consist of the data management layer, in which subscription to changes in service object will be demonstrated, and the developers' portal, in which these interaction will be made easy for the developer to make good use of.

Some examples of the test suite that was developed to test the system as it was implemented, as well as to serve as a regression test suite include:

- Verifying that the underlying communication module can be created
- Verifying that two such components can use the same port
- Verifying that two such components can send messages to each other
- Verifying that two such components can receive respective events
- Verifying that in a configuration of 2 when 1 dies the second one notices it
- Verifying that given the right configuration many members can join a group
- Verify that once a group is established one can start having nodes leave and join and all the existing nodes will obtain the correct view of the current membership
- Verify that both subscribers and publishers can be created
- Verify that a published message arrives to all destination
- Verify that if a message is lost along the way (getting deliberately dropped) it will be re-sent and received eventually (ack-based mode)
- Verify that many publishers and many subscribers can exist and function well simultaneously
- Verify that various subscription and publishing patterns work correctly
- Verify that many topics can operate simultaneously
- Verify that no subscriber, single subscriber, many subscribers and all nodes as subscribers works ok
- Verify that no publisher, single publisher, many publishers and all nodes as publishers works ok
- Verify that publisher and subscriber on the same node works ok
- Verify that creating publisher first or subscriber first works ok
- Verify that processes on different nodes can form a group

- Verify that all capabilities above hold true both for servers and for client nodes.

4 Performance evaluation

The following two tests were conducted:

1. Latency test: 2 clients that were connected to different nodes and each subscribed to a different topic and published 1000 messages to the topic of the other client.
   Each message send and receive time was recorded and an average over all differences between receive and send time was calculated.
   The messages from the clients were sent over WAN lines in which the round trip is 100ms.

   //147.83.42.190:41088: Creating publisher for /147.83.30.133:42663

   //147.83.30.133:42663: Creating publisher for /147.83.42.190:41088

   Starting test for /147.83.42.190:41088

   Starting test for /147.83.30.133:42663

   msgCount: 2000

   Average time: 484

   Figure 4: Latency Test

   If we subtract the round trip of packets from the clients to the node servers (calculated with ICMP echo requests) we get a propagation time of ~ 384 ms.

2. In this test we spawned a varying amount of node servers (that serve client requests) and measured the message delay from the sender to the recipient.

   We spawned 20 clients, and each client subscribed to the topic named after the node server that it connected to, and publishes 1000 messages to all the other topics (that represent the other node servers) – this was done to avoid two clients sending each other messages by having the messages passed in the same JVM and thus skewing the average response time and the minimum response time.

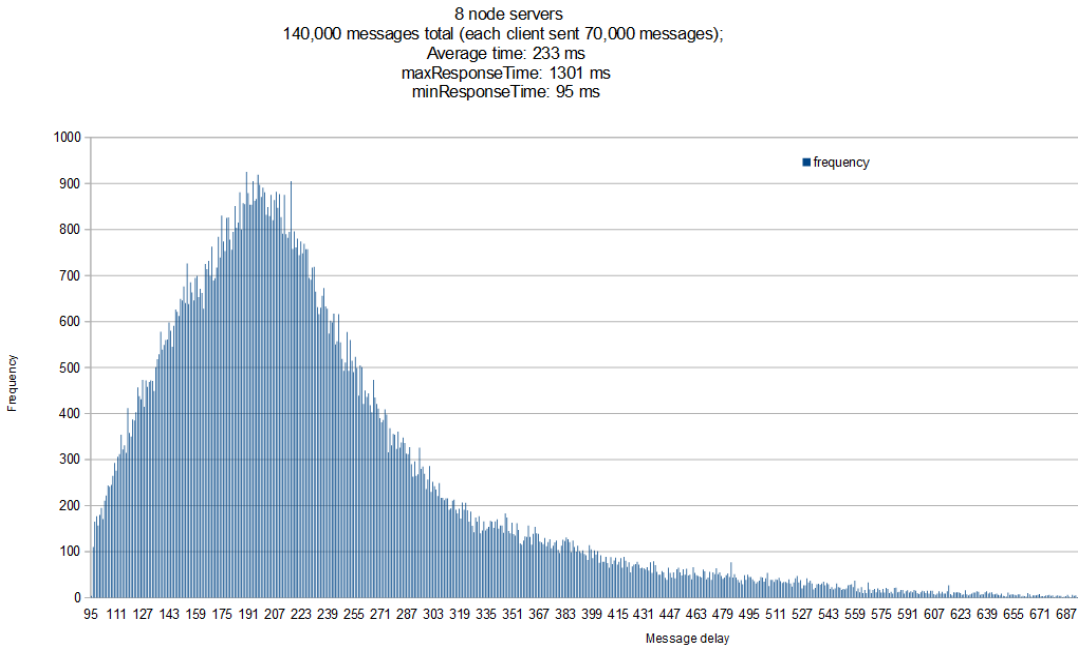


Figure 5: Test Results

As can be seen, the graph has a semi-normal distribution (law of large numbers) and the median is lower than the average. This is due to the communication between the clients and the node servers was being done over WAN lines.

If we compare the graph above with a graph of a test case in which each node sent only 70 messages we see:

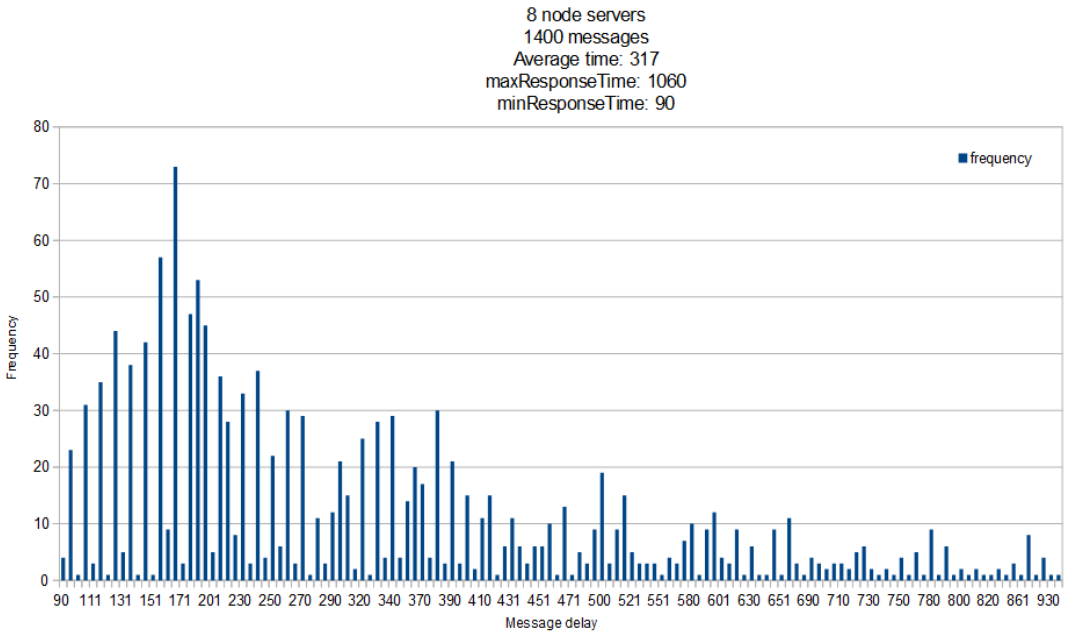
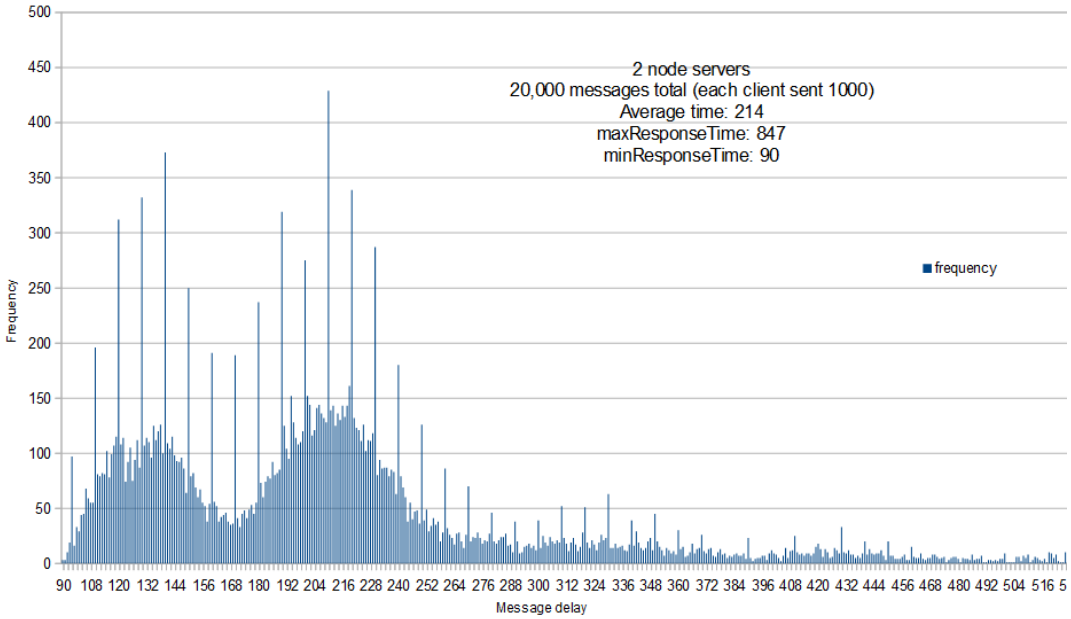


Figure 6: test results

Here, although the number of messages sent was much less, the response time increased by about 50%. This can be explained by the fact that the communication infrastructure does batching of messages instead of sending them one-by-one, which increases throughput under a high workload.



Here in the last test case we ran 2 node servers instead of 8, and we can see that the average time is 214 ms, which is only a bit less than the average time in the test case with 8 node servers with 140,000 messages (233 ms). This demonstrates the linear scalability of the communication infrastructure – we increased the amount of node servers by 4 times but the average time hasn't increased much.

5 External API - for use by 3rd parties

Enclosed is a very concise sub-set of the API that can be used by other components when interacting with the communication infrastructure.

- General node actions
  - create(config) – Will kick-start the process and will join the overlay.
  - close() – An orderly leave of the overlay.
- Membership
  - onMembershipEvent (MembershipEvent) - Obtain membership events by registering a specific call back with the communication sub-system. That callback will be invoked each time the node is made aware of a change in the membership of the group, or in the interests of a group member.
- Pub / Sub
  - createTopic (topicName, config) – Create a specific topic to be used later by publishers and subscribers
  - createTopicPublisher (config, EventListsner, Topic) – Creates a topic publisher instance for a particular topic.

  - createTopicSubscriber (config, EventListener, MessageListener, Topic) - Creates a topic subscriber instance for a particular topic. Messages will be received by the accompanying message listener callback.
  - *publishMessage (OutgoingMessage)* – Method used by the topic publisher to publish a message on its particular topic.
- Messages
  - onMessage (IncomingMessage) – Callback for messages reception

The connection of a 3rd party to the COMPOSE communication infrastructure is done by a client library.

The client library requires only a hostname and port of a coordinator – an entity which is part of the communication infrastructure which redirects the client to a live and running node server.

If the redirection succeeds, the client disconnects from the coordinator and from this point on it has a long lasting TCP connection using the apache Mina library with the node server.

6 Deployment and runtime architecture

The COMPOSE communication infrastructure seed is currently deployed in the Barcelona Supercomputing Center (BSC) server farm in two servers- cloud01 (147.83.30.133) and abiell (147.83.42.190).

Each deployed communications overlay consists of a coordinator and node servers running as Cloud Foundry applications. The node servers autonomously connect to the coordinator upon startup and after restart. When a client wishes to use the communications infrastructure- he contacts the coordinator and is redirected to one of the node servers randomly, for load balancing purposes. If the node server crashes or is unreachable, the coordinator detects this after a grace period and doesn't redirect new clients to it anymore. Clients connected to a server that has crashed will get an exception and the session object they use will become stale. In that case the client needs to create a new session, which will direct it to a different server.

Since provisioning an overlay requires deploying applications to Cloud Foundry, the provisioning is performed asynchronously and on-demand. Each provision attempt of an overlay returns a dormant and unowned overlay which causes a background provisioning of a new overlay for a subsequent request.

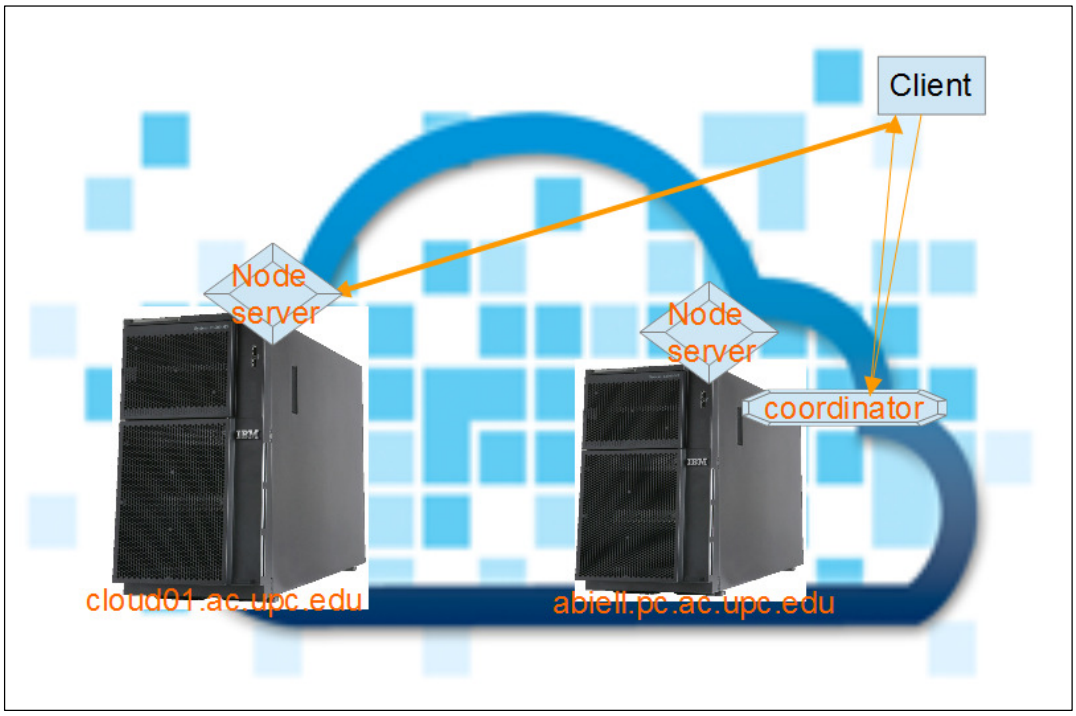


Figure 7: Deployment of the communications components

7 Integration with the Developers' Portal

One of the main manners in which this component is used within COMPOSE is to connect different layers by passing information from one to the other. A COMPOSE application, created by using the Developers' Portal, can subscribe to updates on a specific service object. In order to perform a subscription, first the service object in question needs to be located within COMPOSE (see Figure 7), and then the desired service object information needs to be selected, and the node needs to be specified to subscribe to update events (see Figure 8).

After the node is configured, it emits the value of the property of the service object upon a value change in the service object itself. Thus, the application that registered its interest will receive the new value of the service object whenever there's a change.

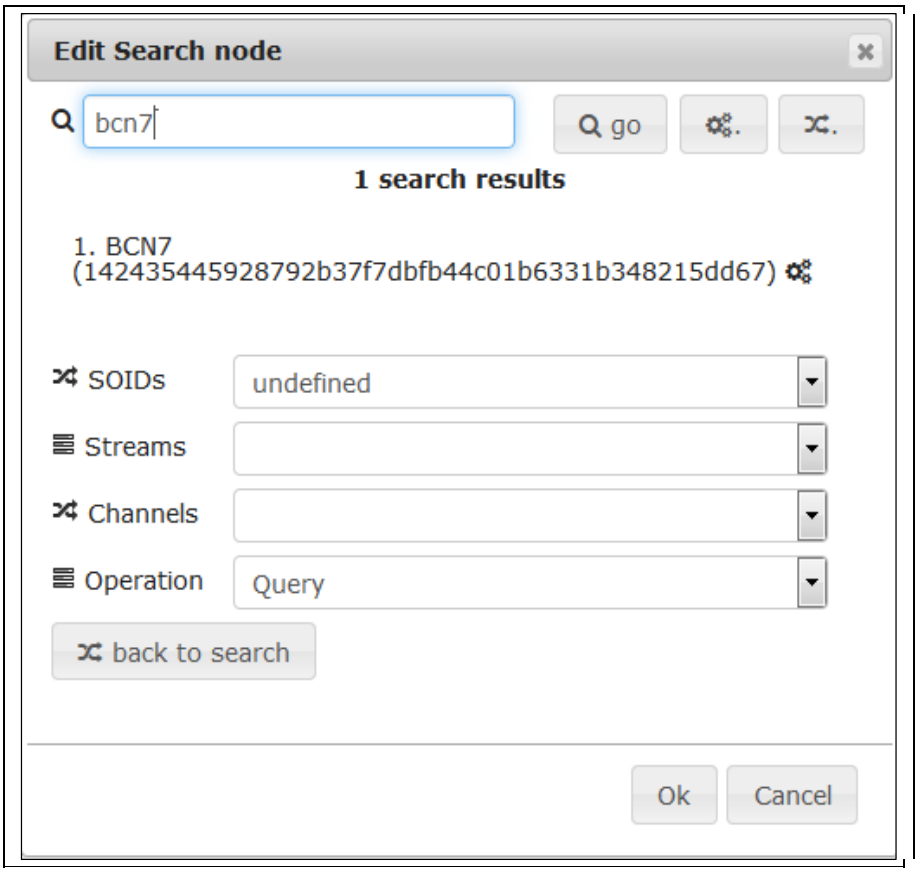


Figure 8: Locating the service object of interest

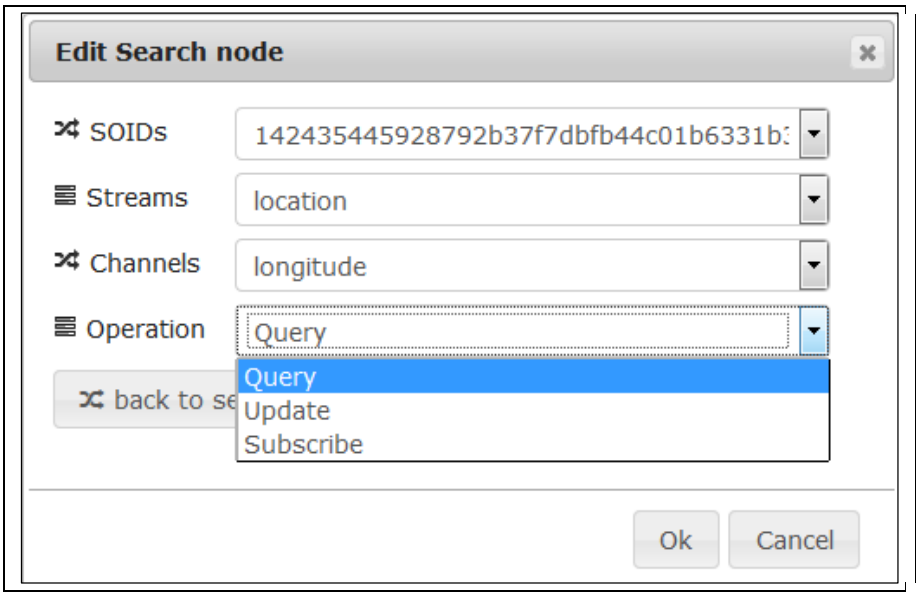


Figure 9: Subscribing to change notifications

Figure 10: The resulting node in the palette

The resulting NodeRed node in the Developers' Portal (see Figure 10) uses a client library implemented in JavaScript to connect to ServioTicy in order to receive updates.

Upon invocation of the application's service object NodeRed node in subscription mode, it sends an HTTP POST request to ServioTicy and asks to subscribe to the topic with the following name: '/' + serviceObjectId + '/streams/' + SteamId + '/subscriptions'. That would effectively create the subscription of the application interest within ServioTicy.

The POST also includes the future topic string it expects ServioTicy to use when sending updates to the applications that registered their interest.

After the subscription request, the node starts listening for update requests by using the JavaScript implementation of the client library and subscribing to the topic sent by the POST request. Upon reception of a message the NodeRed node forwards each message to the next application node in the flow.

ServioTicy publishes a data change event through the COMPOSE communication infrastructure, which propagates the event to the subscription node component in the COMPOSE applications. Internally this capability uses the communication infrastructure scalable pub / sub mechanism to disseminate information between the service objects layer and the COMPOSE applications layer.

8 Future directions

- Scalability – Scalability enhancements will be assessed along with the progress of the various parts of the project. Sound design plans are already available for a hierarchical approach to the overlays structure.
  Additional scalability enhancements may be introduced based on findings of specific issues with the currently demonstrated system. For example, enhancements to the attribute replication and gossip mechanisms, or considering NACK-based approaches for messages dissemination reliability.
- Point-to-point (overlay or direct) – Exposing the possibility to transmit messages to a particular destination. Upon need two flavours of a point-to-point mechanism can be supported. First, routing messages over the existing overlay links. Second, creating specifically tailored connections to a particular destination in addition to the existing overlay links, in case heavy traffic is expected between the two nodes (such as if there is a need to support data replication between the 2 nodes).
- Broadcast – Sometimes there's a need to transmit a message to all members of the group. This is mostly the case for administrative purposes. An efficient manner in which to perform broadcast at scale in an overlay system is important especially if this mechanism is to be used extensively and gets more important as the scale gets bigger.

We have designed such a service that enables efficient broadcast by halving the target range at each hop of the message, and may implement and expose it upon need.

- Bulletin board semantics– a Bulletin board can be thought of as an implementation of a shared state (or shared memory). Nodes get notified of the last value submitted on a specific topic but there's no guarantee that all values will be delivered (intermediate values may be skipped). This capability is often associated with maintaining distributed monitoring information in an efficient manner. This capability can be built using the existing pub / sub design and implementation and will be considered based upon the platform's needs.