PADRES Demo Guide

PADRES Developer Team

The PADRES release includes number of demo applications to demonstrate the capabilities of PADRES. This include the followings:

- **Stock Quotes**: This demonstrates a stock quote application where a number of subscribers subscribe to stock data published by a number of publishers.
- **Fixed Routing in Cyclic Overlays**: This demo shows how a fixed routing path is created despite of cycles in the overlay.
- **Dynamic Routing in Cyclic Overlays**: This shows how PADRES exploits the multiple paths between two end points in a cyclic network to perform dynamic routing in case of failures.
- **Failure Detection**: This demonstrates the ability of the PADRES to detect failures.
- **Web Client**: This demonstrates the web client interface connected to the overlay topology.

1 Stock Quote Demo

1.1 Components and parameters

StockPublisher: The publisher client connects to the specified broker and advertises a given stock symbol. The publisher then emits a publication at a fixed rate.

-b The URI of the broker to connect to. (i.e. rmi://localhost:1099/Broker1)

-i The ID of the client. Please ensure that IDs are **unique** with respect to a single edge broker. In other words, clients (publishers and subscribers alike) connected to the same broker must have a different ID. (Default: an ID is generated using current time)

-s The stock symbols this publisher will issue. The symbols must be present in the directory "/demo/data/stockquotes". To issue multiple symbols, the symbols must be seperated by /. (i.e. ANTP/AMZN)

-d The delay, in seconds, after the advertisement is issued before publications start. A long delay is necessary in large topology to ensure the advertisement is fully propagated. (Default: 0)

-r Rate of publication, in number of messages per minute. If multiple symbols are specified, the rate for each symbol must be set and separated by /. (i.e. 3/5)

StockSubscriber: The subscriber client connects to the specified broker and submits the provided subscription. Received publications are logged using *log4j*.

-b The URI of the broker to connect to. (i.e. rmi://localhost:1099/Broker1)

-i The ID of the client. Please ensure that IDs are **unique** with respect to a single edge broker. In other words, clients (publishers and subscribers alike) connected to the same broker must have a different ID. (Default: an ID is generated using current time)

-s The subscription to be issued. See Section 1.2 for examples.

1.2 Format

Advertisements issued by the publishers are of the form:

[class,eq,'STOCK'],[symbol,isPresent,'A'],[open,isPresent,1],[high,isPresent,1],[low,isPresent,1],
[close,isPresent,1],[volume,isPresent,1],[date,isPresent,'A']

Publications are then read from the symbol's associated data file.

Example subscriptions

- Subscribe to all publications: [class,eq,'STOCK']
- Subscribe to AMZN: [class,eq,'STOCK'],[symbol,eq,'AMZN']
- Subscribe to stocks with volume higher than 5: [class,eq,'STOCK'],[volume,>,5]

2 Running the demo

1. Start your brokers in any desired topology.
2. Connect your publishers and subscribers in any given order to brokers. Use "demo/bin/stockquote/startSQsubscriber.sh" and "demo/bin/stockquote/startSQsubscriber.sh" respectively.

Example commands

```
demo/bin/stockquote/startSQsubscriber.sh -i Client1 -s "[class,eq,'STOCK']" -b rmi://localhost:1099/Broker1
demo/bin/stockquote/startSQpublisher.sh -i Pub1 -s ANTP -r 15 -d 15 -b rmi://localhost:1099/Broker1
```

This will start a publisher and subscriber connected to the same broker.

3 Fixed Routing in a Cyclic Overlay

3.1 Topology

There are four brokers A, B, C, and D running at ports 1099,1098,1097 and 1096 respectively. Three swingClients are connected to the overlay; the first two connecting to Broker A, while the last connected to the broker D. The overlay is arranged as shown in Figure 3.1. The neighborhood relationship among the brokers creates a cyclic overlay.

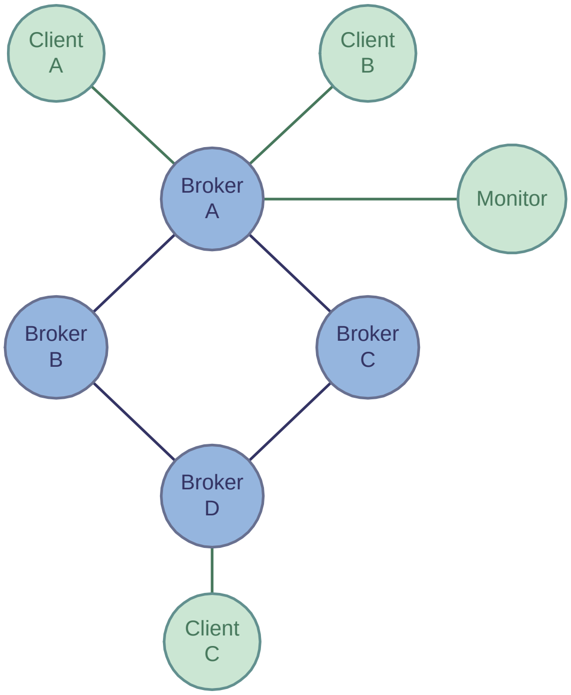

Figure 1: The overlay topology for the demo.

3.2 Hack for the Fixed Cycles Demo

For fixed cycle demo: There are some problems for what the monitor display on the screen. Sometimes, the monitor could not display correct information of the network, please see the following two scenarios.

1. If one broker (say B) is stopped, in addition to this broker is marked in the display as failure, some other brokers (e.g., A and C) will be detected as failure as well. That is because the HEARTBEAT_REQ and HEARTBEAT_ACK advertisement tree of these brokers are setup by passing brokerB. The corresponding HEARTBEAT publication messages, therefore, could not be routed through brokerB, since it is stopped already. The broker who could not receive the ACK msg from its neighbours will detect neighbour's failure.

2. The monitor could not display how many messages are routed on the link either for a similar the reason. If the TRACEROUTE notification message is routed by passing a failure broker, then this msg will not arrive at the monitor. Consequently, the monitor could display it on the screen. A similar issue applies to BROKER_INFO msg too.

```
SOLUTION FOR FIXED CYCLE DEMO: A temporary solution for demo is applied now
to still let these passing messages route through the failure broker. Some
hard code are added in the InputqueueHandler.java.
```

3.3 Running the Demo

1. **Setup the Network Topology**
   (a) run `demo/bin/fixedcycles/demo_fixedcycles.sh` : this will start the four brokers and three clients with a monitor
   (b) verify the topology is set successfully
      i. in the monitor, click *Main* → *Connect to Federation*.
      ii. fill in Hostname and Port to connect to BrokerA, i.e., localhost and 1099.
      iii. click OK.
      iv. wait for a few seconds for the topology to appear on the monitor pan. Click *Main* → *Apply Layout*, if it fails to appear.
      v. verify that the four brokers and four clients (one is the monitor itself) are there.
2. **Prepare for Demonstartion**
   (a) in the monitor, highlight the broker A
   (b) click *Broker* → *Trace by ID*
   (c) fill in the Trace ID with "CYCLE_DEMO", where CYCLE_DEMO is the class name of the publication we are going to disseminate.
   (d) click OK.
   (e) click *Main* → *Show Edge Message Counter*: this will enable displaying the packet counts along the links. Initial values of zeros will be shown now.
   (f) (optional) click *Main* → *Edge Throughput Indicator* → *On*: this will change the line width of the edges between brokers as more packets are pushed through them.
3. **Demonstration of Publish/Subscribe in the Cyclic Network** [ClientA publisher/ClientC subscriber]
   (a) Advertise:
      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoFixedCyclesAdv.txt` , located at `demo/data/fixedcycles/`
      iii. click Open.
   (b) Subscribe:
      i. in the clientC, click *Application* → *Deployment*
      ii. choose `DemoFixedCyclesSub.txt` , located at `demo/data/fixedcycles/`
      iii. click Open.
   (c) Publish:
      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoFixedCyclesPub.txt` , located at `demo/data/fixedcycles/`
      iii. click Open.
   (d) Observe result:
      - You should be able to see the route A→B→D or A→C→D is changed to a dark color, and message counter on the link is changed to 1 in the monitor screen. The highlighted path shows the route taken by the subscription to reach the publisher. It also shows that the publication takes a non-cyclic path in the cyclic overlay.
4. **Simulate a Failure in the Overlay**
   (a) suppose that clientA choose the route A→B→D to publish, click the broker B in the monitor.
   (b) click *Broker* → *Stop Broker*

(c) click OK.

   - After a while, both brokerA and brokerD will detect the failure from brokerB, so that brokerB, the link between brokerA and B, and the link between brokerB and D turn red indicating failure.

5. **Publish/Subscribe in the Failed Network**

   (a) Now advertise and publish through Client B as described before.

   - Note that subscription sent by ClientC is already received by CleintB and active.
   - After this, you should be able to see the other (only available) route A→C→D is changed to a dark color, and message counter on the path is changed to 1 in the monitor screen. This means that the new publisher can pick another route in the cyclic network to publish.

4 Dynamic Routing with Cyclic Overlays

4.1 Topology

There are four brokers A, B, C, and D running at ports 1099,1098,1097 and 1096 respectively. Three swingClients are connected to the overlay; the first connecting to Broker A, and the other connecting to the broker D. The overlay is arranged as shown in Figure 4.1. The neighborhood relationship among the brokers creates a cyclic overlay.

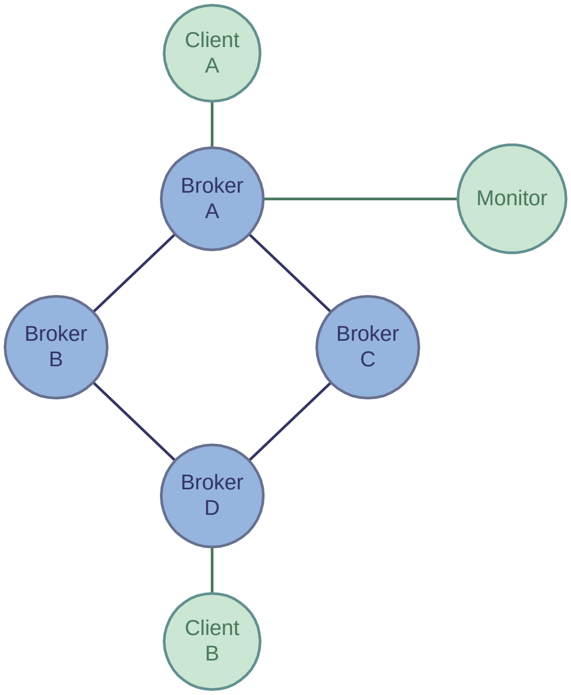

Figure 2: The overlay topology for the demo.

4.2 Running the Demo

1. **Setup the network topology**
   (a) Run `demo/bin/dynamiccycles/dynamiccycles.sh` : this will start the four brokers, two clients, and the monitor.
   (b) Verify the topology is set successfully
      i. in the monitor, click *Main* → *Connect to Federation*.
      ii. fill in Hostname and Port to connect the monitor to BrokerA. i.e. localhost and 1099.
      iii. click OK.
      iv. Wait for a few seconds. If the display fails to show the topology, click *Main* → *Apply Layout* to refresh the topology.
      v. Verify four brokers and three clients are there.
2. **Prepare for demonstration**
   (a) In the monitor, highlight the broker A
   (b) click *Broker* → *Trace by ID*
   (c) fill in the Trace ID with "CYCLE_DEMO", where CYCLE_DEMO is the class name of the messages used.
   (d) click OK.

(e) click *Main* → *Show Edge Message Counter*. This will enable displaying the packet counts along the links. Initial values of zeros will be shown now.

(f) click *Main* → *Edge Throughput Indicator* → *On*. This will change the line width of the edges between brokers as more packets are pushed through them.

3. **Subscribe:** (Client B is subscriber)

   (a) in the clientB, click *Application* → *Deployment*

   (b) choose `DemoDynamicCyclesSub.txt` located in `demo/data/dynamiccycles` .

   (c) click Open.

4. **Demonstration of publication routing in cyclic network:** (Client A is publisher)

   (a) Advertise (Adv1)

      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoDynamicCyclesAdv1.txt` located in `demo/data/dynamiccycles`
      iii. click Open.

   (b) Publish:

      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoDynamicCyclesPub.txt` located in `demo/data/dynamiccycles`
      iii. click Open.

   (c) Observation:

      - A path from Broker A to D should be highlighted in the monitor with the message count of 1. Let's assume it A→B→D for the rest of the demo.

5. **Demonstration of dynamic routing of publication messages in a cyclic overlay** (Client A is publisher, Client B is subscriber)

   (a) If the observed publication message path above is A→B→D, stop Broker B

      i. in the monitor, highlight the broker B
      ii. click *Broker* → *Stop Broker...*
      iii. click OK
      iv. Wait until both brokerA and brokerD detect the failure of brokerB and, therefore, brokerB, the link between brokerA and B, and the link between brokerB and D are marked as failed (red).

   (b) Advertise with brokerB down (Adv2) (this ensures adv2 is routed along A→C→D)

      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoDynamicCyclesAdv2.txt` located in `demo/data/dynamiccycles`
      iii. click Open.

   (c) Publish for Adv1 with BrokerB still down (it should find another way, A→C→D, to route the message in the cyclic network, thus proving dynamic routing.)

      i. in the clientA, click *Application* → *Deployment*
      ii. choose `DemoDynamicCyclesPub.txt` located in `demo/data/dynamiccycles`
      iii. click Open.

   (d) Observation:

      - you can see the route A→C→D is changed to a dark color and message counts on the links are changed to 1 in the monitor screen. This means that the publication, which does not conformed by adv2, is routed along A→C→D in the cyclic network.

5 Failure Detection

In this demo we exercise the failure detection feature of PADRES. Brokers publish and subscribe to heartbeat messages and upon missing a set number of heartbeat notifications (i.e., publications), a broker signals failure. We demonstrate this feature through a client monitor that depicts the failure in a graphical user interface.

5.1 Topology

The following topology is used.

```
Topology: topology_fd

There are four brokers. BrokerA, BrokerB, BrokerC, and BrokerD are running at ports 1099,1098,1097 and 1096, respectively.
There are two swingClients. Client1 is connected to BrokerA, and Client2 is connected to the BrokerC.
The monitor is connected to BrokerA.

          Client1
             |
          BrokerA --------- Monitor
             |
          BrokerB
             |
          BrokerC --------- Client2
             |
          BrokerD
```

5.2 Running the Demo

```
1. Set up the topology.
 a. Start the 4 brokers.
    ./bin/startbroker -uri socket://localhost:1099/BrokerA
    ./bin/startbroker -uri socket://localhost:1100/BrokerB -n socket://localhost:1099/BrokerA
    ./bin/startbroker -uri socket://localhost:1101/BrokerC -n socket://localhost:1100/BrokerB
    ./bin/startbroker -uri socket://localhost:1102/BrokerD -n socket://localhost:1101/BrokerC
 b. Start and connect the clients.
    ./bin/startclient -b socket://localhost:1099/BrokerA -i Client1
    ./bin/startclient -b socket://localhost:1101/BrokerC -i Client2
 C. Start the monitor.
    ./bin/startmonitor
 d. Connect the monitor to BrokerA.
    Main -> Connect to Federation -> Enter socket://localhost:1099/BrokerA

2. Set up a publish/subscribe interaction.
 a. Send the following advertisement from Client1.
    a [class,eq,'temp'],[area,eq,'tor'],[value,>,0]
 b. Send the following subscription from Client2.
    s [class,eq,'temp'],[area,eq,'tor'],[value,>,0]
 c. Send the following publication from Client1.
    p [class,'temp'],[area,'tor'],[value,10]
 d. Verify that the publication is received at Client2 (should be displayed).
 e. From the monitor, select Main -> Global Failure Detection -> OK (it should be enabled).
3. Disable an edge broker.
 a. Kill the process that runs BrokerD
 b. After several seconds, BrokerD should become red.
```

c. Resend the publication from Step 2c from Client1 and verify it is received by Client2.

4. Disable a non-edge broker.
   a. Kill the process that runs BrokerB
   b. After several seconds, BrokerB should become red.
   c. Resend the publication from Step 2c from Client1 and verify that Client2 does not receive the message.

6 Web Client

6.1 Demo purpose

This demo shows the Web user interface for interacting with PADRES. In this demo, we carry out the following steps:

1. Start a broker
2. Start two separate Web clients, publisher and subscriber, from a Web browser
3. Connect to the broker from the Web clients
4. From the publisher, issue advertisement
5. From the subscriber, issue subscription
6. From the publisher, issue publications

6.2 Demo operation

1. Start a broker on localhost at port ”1099” with ID ”Broker1” 1. Run ./bin/startbroker -uri socket://localhost:1099/Broker1 2. Start publisher client on localhost at port ”8080” with ID ”Publisher” 1. Run ./bin/startclient -i Publisher -web -c ./demo/etc/web/client/client1.properties 2. Start a browser window and type http://localhost:8099 on the address bar. 3. Start a publisher client on localhost at port ”8090” with ID ”Subscriber” 1. Run ./bin/startclient -i Subscriber -web -c ./demo/etc/web/client/client2.properties 2. Start another browser window and type http://localhost:8100 in the address bar. 4. Connect to ”Broker1” and start publishing and subscribing 1. From ”Publisher” and ”Subscriber”, press ”Connect” button and enter the URI of ”Broker1” 2. From ”Publisher” issue an advertisement: a [class,eq,’web’],[price,¿,0] in the user input field. 3. From ”Subscriber” issue a subscription: s [class,eq,’web’],[price,¿,10] in the user input field. 4. From ”Publisher” issue an publication: p [class,’web’],[price,15] in the user input field. 5. ”Subscriber” should receive a notification of the publication.

6.3 Remarks

1. When starting a Web client, a Web server on the client side is started behind the scene.
2. The port_number has to be unique for each client.
3. Default configuration for the Web client is in /etc/web/client.