



**IP-FP6-015964**

**AEOLUS**

**Algorithmic Principles for Building Efficient Overlay Computers**

**Deliverable D6.2.12**

**Software for Dynamic Replication in Global Overlay Computers**

---

Responsible Partner: Max-Planck Institute for Informatics (D)  
Report Preparation Date: February 2010

Contract Start Date: 01/09/05 Duration: 54 months  
Project Co-ordinator: University of Patras (EL)

This deliverable consists of the software for the dynamic replication of data in global overlay computers. It is based on the P2R2 method for near-optimal dynamic replication in global overlays [1]. The method itself is part of Deliverables D3.1.3 and D3.1.4 [2, 3]. The earlier deliverable D6.2.9 [4] had been based on a stand-alone implementation in C++ and was geared for a network simulation. The current deliverable provides a complete re-implementation of the P2R2 method. The software is written in Java and uses the JXTA-based interfaces of the AEOLUS testbed, on which the software has been tested. Details for installing and running the software are provided in the Appendix.

Replicating data in distributed systems is often needed for availability and performance. In unstructured peer-to-peer networks, with epidemic messaging for query routing, replicating popular data items is also crucial to ensure high probability of finding the data within a bounded search distance from the requestor. The P2R2 method considers such networks and aims to maximize the probability of successful search. Prior work along these lines has analyzed the optimal degrees of replication for data items with non-uniform but global request rates, but did not address the issue of where replicas should be placed and was very limited in the capabilities for handling heterogeneity and dynamics of network and workload. The P2R2 algorithm for dynamic replication addresses all these issues, and determines both the degrees of replication and the placement of the replicas in a provably near-optimal way. P2R2 can provably guarantee a successful-search probability that is within a factor of 2 of the optimal solution. The algorithm is efficient and can handle workload evolution. Simulations with realistic network configurations and workloads and measurements in the real network of the AEOLUS testbed have shown that the convergence rate is fast and that P2R2 outperforms prior methods by a large margin.

## References

- [1] Mauro Sozio, Thomas Neumann, Gerhard Weikum: Near-optimal dynamic replication in unstructured peer-to-peer networks. ACM Symposium on Principles of Database Systems (PODS), Vancouver, Canada, 2008.
- [2] AEOLUS Deliverable D3.1.3: Algorithms and Techniques for Data Services with QoS Guarantees, August 2008.
- [3] AEOLUS Deliverable D3.1.4: Algorithms and Techniques for Data Services with QoS Guarantees, February 2010.
- [4] AEOLUS Deliverable D6.2.9: Software for Dynamic Replication in Global Overlay Computers, August 2008.

## A Instructions for installing and running the p2r2-software

### A.1 Setup

Once you've gotten this code, you will need the following components:

```
ant 1.7.1 or later
java 1.6 or later
```

Also, only unix-based (mac os included) systems are supported for the server side due to the use of shell scripts.

Once you've installed those, make sure you also set the following environment variables:

```
JAVA_HOME - pointing to your java folder, such that $JAVA_HOME/bin/javac is
            found
ANT_HOME - such that $ANT_HOME/bin/ant resolves to the ant build system
```

Once these two have been set, you can execute the following ant projects

- all (this is the default project)  
this project will build the project and upload the needed files to the aeolus framework (such as p2r2.jar and config files)
- local  
builds the project, does not perform upload
- usage  
prints help on the available projects

Ant uses the following (default) folders for its output (which can be configured in the build.properties file)

- build  
this is a scratch folder that will contain temporary state information
- dist  
contains the generated p2r2.jar

You don't need to update any of the values provided in build.properties, the defaults will work fine as long as you have set up properly ant and java.

### A.2 Configuration

The p2r2 implementation program can be configured via the configs/app.properties, the following are the recommended values to update:

## CLIENT SIDE PROPERTIES

- `query.wait.time`: specifies how long (milliseconds) a peer will wait for a query to be discarded.
- `query.concurrence`: specifies how many concurrent queries can a peer send
- `query.ttl`: specifies query time to live (hops)
- `zipf.alpha`: clients will generate a frequency table based on a zipf distribution. the alpha value of that distribution is configured through this value
- `peer.sleepbeforefirst.min/max`: specifies the time (milliseconds) before a peer will wait before send its first query, this is in order to allow for set-up time
- `peer.capacity.min/max`: specifies the storage capacity of a peer, which will be a value in the [min, max] interval
- `update.step`: peers generate messages to the server. this parameter configures how often (measured by sent queries) they send update information

## ITEM DOWNLOAD SIMULATION PROPERTIES

- `item.download.bitrate`: bitrate at which files are downloaded (this is to simulate item download)
- `item.search.time.min/max`: time to wait before starting a download of a file actual time will be in the [min, max]
- `item.find.probability`: after search time has elapsed, items have a probability of finding a file

## SERVER SIDE PROPERTIES

- `topology.size`: number of nodes in the network
- `topology.edge.probability`: specifies the probability of an edge to exist between two nodes
- `item.total`: specifies the total number of items that exist in the network
- `item.maxsize` and `item.minsize`: specify the minimum and maximum size that

an item will have. the value for each item will be [item.minsize, item.maxsize]

- report.output.folder: path where csv reports are written
- graphml.output.folder: path where topology files will be saved

Two additional configuration files exist, and it is highly recommended not to modify them. They are configs/peer-services.txt and configs/server-services.txt. They specify the services (jxta applications) that clients/servers will execute.

### A.3 Execution

Once everything has been set-up and configured, there are two scripts to be used in order to start the application on the server side (client-side is managed by aeolus).

- generate-instances.sh

generates clients / server instances on a local machine, accepts the following command line parameters

- + <number-of-instances> - how many clients will be run on the machine where this script is invoked
- + <jxta-port-start> - network port to use. if clients are also run, each instance will have its own port. say you want to run 20 clients and a server on a machine, and you specify 9700 as a value for this parameter, this will generate clients using ports 9700, 9701, ... 9719 and a server on port 9720
- + <local-name> - name to be prepended to the ip address of the local computer, this is used to identify peers
- + <start-server> - specifies whether a server will be started or not

- stop-instances.sh

does not take any parameters. it stops the instances run on the local machine

A typical run of the implementation would be:

```
your-shell$ ant
your-shell$ ./generate-instances 0 9700 my-computer true
```

This will:

- 1) build the project and upload the needed files to the aeolus framework
- 2) generate a server using the "my-computer" id on port 9700

The aeolus-instances/server and aeolus-instances/node\_N contain the stdout.log and stderr.log files that can be tailed in order to analyze problems.

On the server side a UI interface will be displayed, showing charts and a control console.