

# ***JORAM 3.5***

## ***PERFORMANCES IMPROVEMENT***

---

*Author: Frédéric Maistre.  
June 2003, 20'*

# 1. Introduction

The 3.5 release includes a new management of message persistence. In the previous Joram versions, messages were persisted within the general state of the entity which hosted them. For example, messages on a queue were persisted because the queue itself was persisted, and because they were part of it.

This implementation made it impossible to individually control the persistence of the messages. When running a Joram platform in persistent mode, all messages were persisted, even if they were sent with the option `javax.jms.DeliveryMode` set to `NON_PERSISTENT`.

In the 3.5 version messages are not part anymore of the state of the entity which hosts them. Message persistence is controlled individually, the option `javax.jms.DeliveryMode` is now taken into account.

**Why this change?** A performance issue has been for long identified, specially when running Joram in persistent mode. Particularly, when a message producer kept sending messages, the server quickly became overloaded and consumers did receive messages at a very low rate. The first idea was to address this issue. But also, the modifications which occurred to the code have been driven by the requirement of providing some flexibility to the persistence of the messages, for example by allowing in the future messages to be saved in a database rather than as files on disk.

## 2. Measures

The test clients used for measuring Joram's performances are available in the samples distribution, in the `perfs` package. Their use is documented in the JORAM 3.5 samples guide.

The worst situation for a consumer has been taken into account. A message producer keeps sending messages while the consumer tries to consume them. This situation is bad because the constant sending of messages tends to overload the server and totally stop the messages flow to the consumer. Both Point-To-Point and Publish/Subscribe communication modes have been tested.

Performances depend on many parameters. One of them is the transactional nature of the applications' sessions. When producing messages from a non transacted session, messages are sent one by one. When producing messages from a transacted session, messages are sent all together when committing the session. Similarly, when receiving the messages on a non transacted session, messages are acknowledged one by one. When receiving them on a transacted session, messages are acknowledged all together when committing the session.

This leads to very different behaviours, as shown by the following pictures.

Joram 3.5 also differently processes the messages persistence according to the nature of the consuming subscriber (durable, or not). That's why the performances of the subscriber of the 3.4 release are compared with the performances of the durable and the non durable subscribers of the 3.5 release.

The results shown in the next sections are the messages mean travel time. While the producer constantly saturates the server with messages, the consumer keeps consuming. For each group of 10 (Point-To-Point tests) or 50 (Pub/Sub tests) messages it receives the mean travel time is computed. This for showing how this travel time evolves, and for showing any overflow situation.

The sent messages are bytes messages each carrying an array of 1024 bytes.

The tests were runned with joram 3.4.2 and joram 3.5.0, using jdk1.4.1\_01 on a Win 2K environment. Machine is a 1.1GHz Pentium, with 256Mo of RAM and SCSI drive.

### 3. Point-to-Point mode

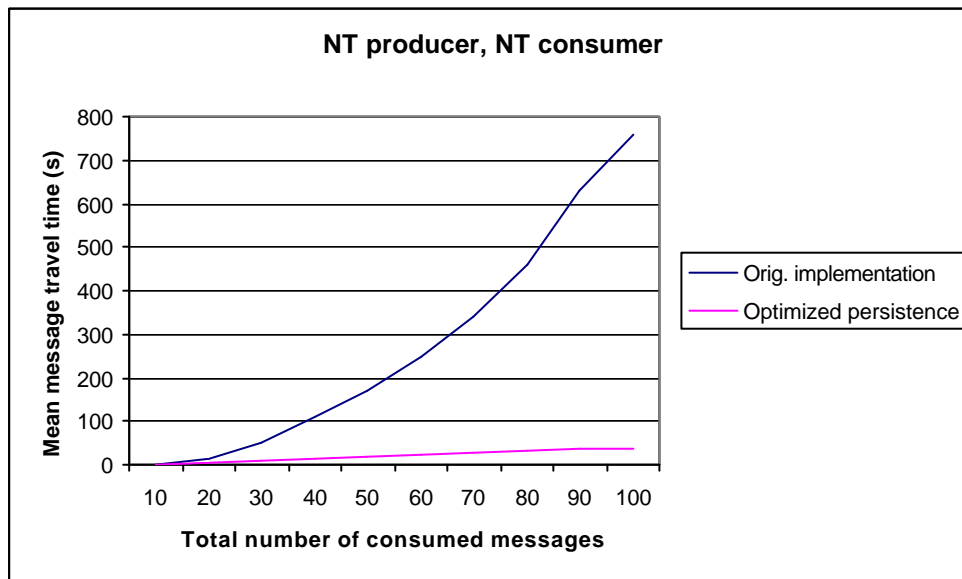
The following pictures clearly show Joram 3.4's problem of saturation. While the number of produced messages increases, the flow of consumed messages decreases (as shown by the fact that the messages travel time increases).

The worst results are obtained with a transacted producer. Groups of 10 messages are constantly sent, multiplying by 10 the flow of incoming messages on the server. This case is the most favorable to saturation.

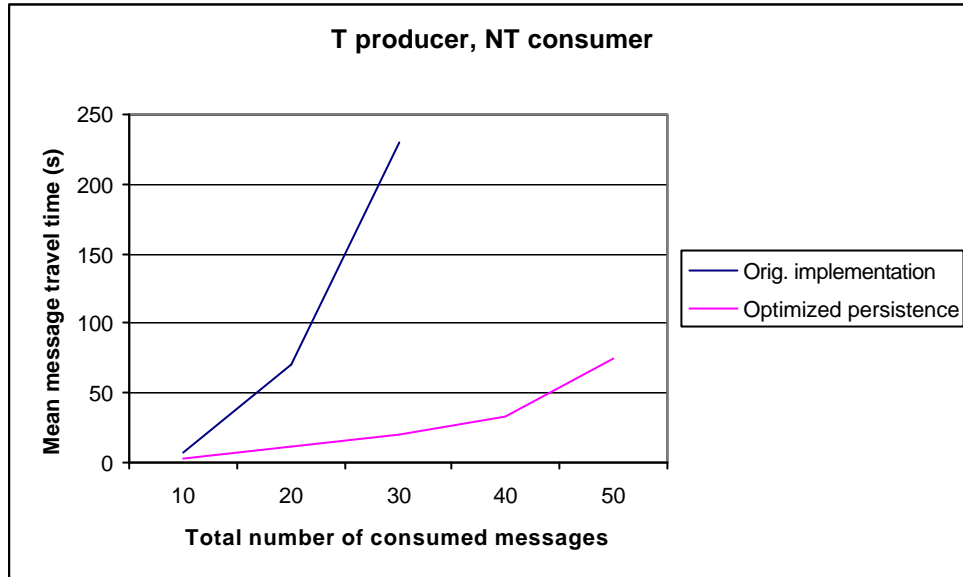
A transacted consumer works to the contrary better. By grouping its acknowledgements, it decreases its traffic with the server generated by the sending of the acknowledgements.

The 3.5 release obviously works better. The travel time growth looks linear rather than exponential, as in the 3.4 case. Total saturation has not been reached during these tests, contrary to the 3.4 release.

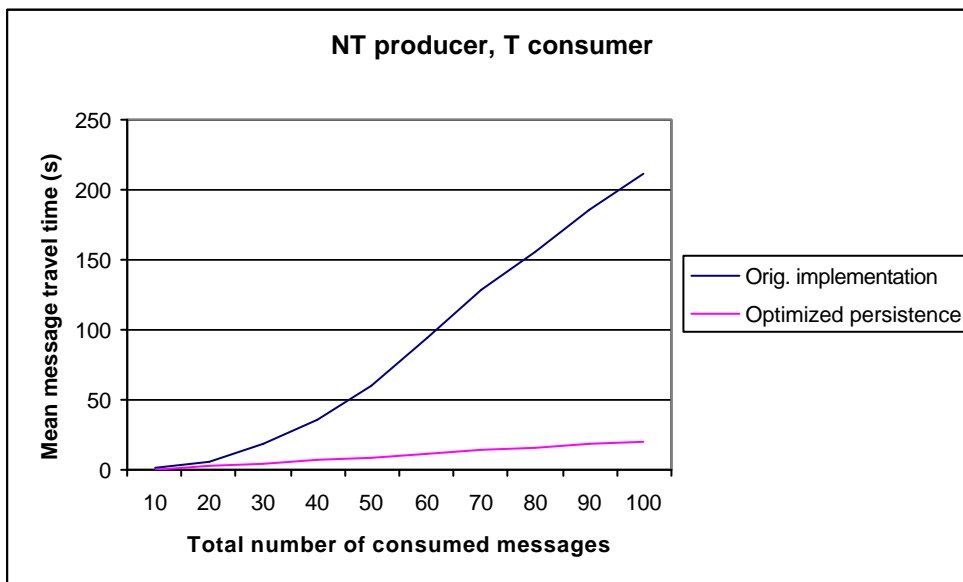
#### Case 1: non transacted producer and consumer



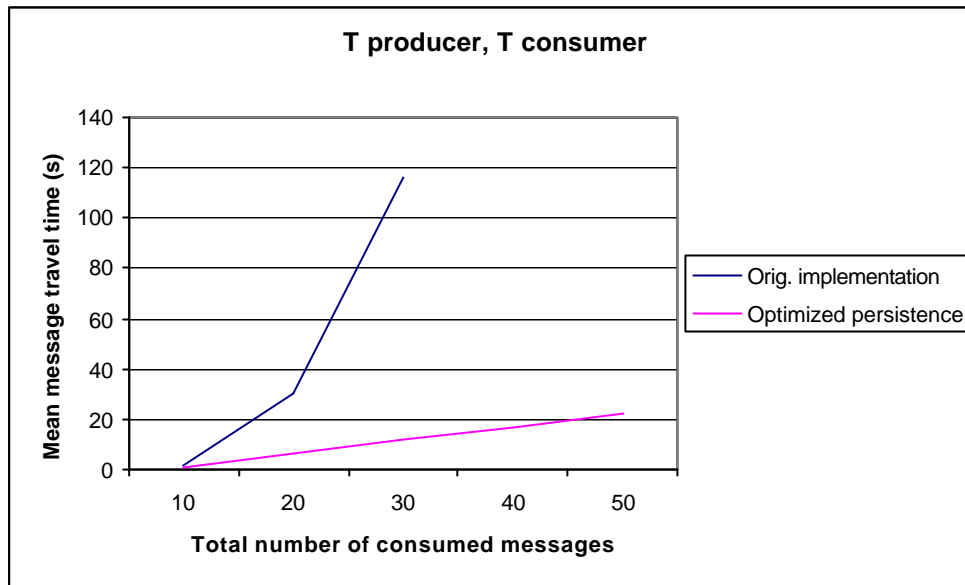
Case 2 : transacted producer, non transacted consumer



Case 3 : non transacted producer, transacted consumer



Case 4: transacted producer and consumer



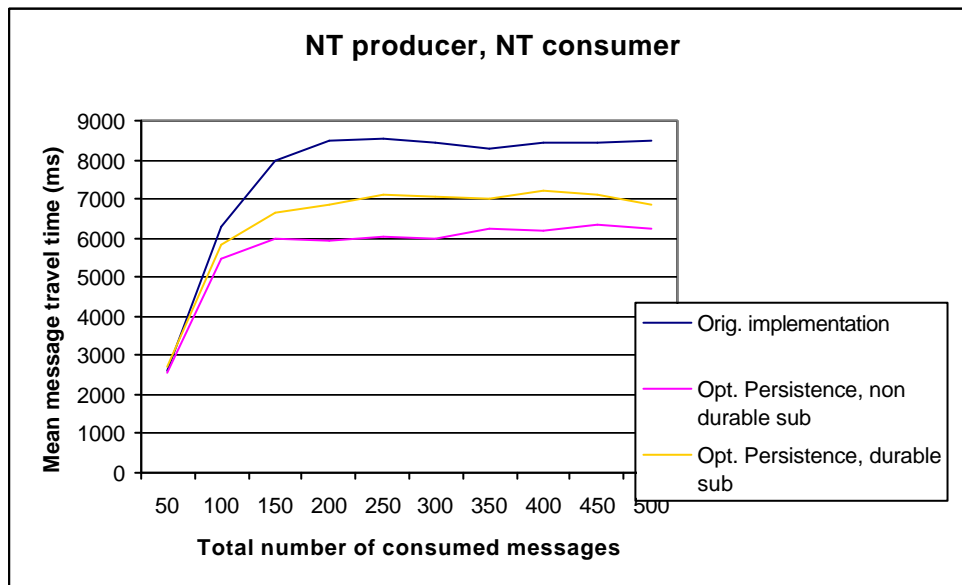
## 4. Publish/Subscribe mode

The results obtained when testing the Pub/Sub mode are totally different from those obtained when testing the PTP mode.

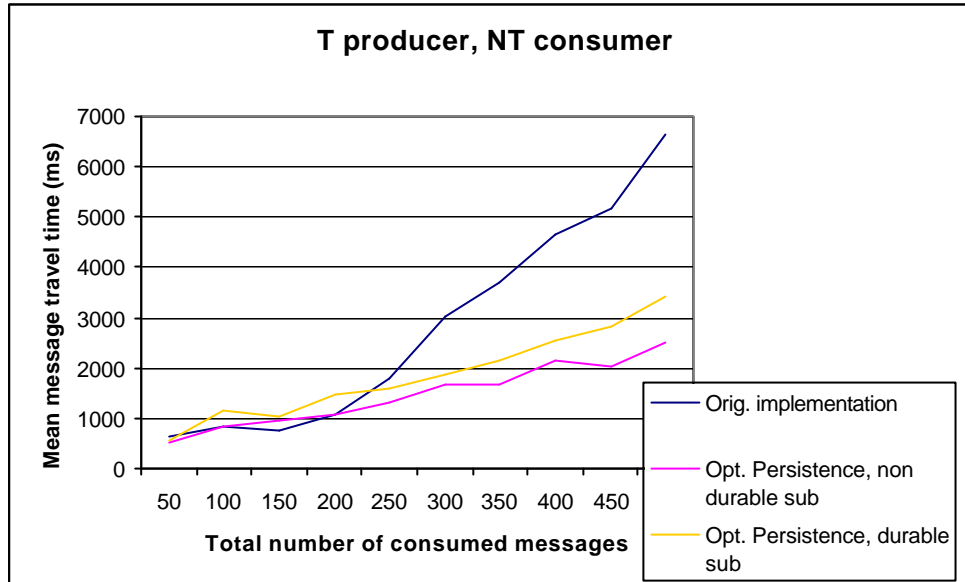
First of all, the messages travel time is much lower in this case. Saturation may still occur (specially, once again, when using a transacted producer), but much later. This is due to the totally different semantics of Pub/Sub communication compared to PTP communication. Whereas in the PTP case each message is consumed by a single application, on request, in the Pub/Sub case each message is consumed by all the subscribers. Pub/Sub mode is a push mode, when a message (or a group of messages) reaches the topic, it is automatically forwarded to the subscribers. In the PTP mode, when a message reaches the queue, it is stored on the queue, and delivered if a consumer requests it (pull mode).

Improvements are then less dramatic for the Pub/Sub mode than for the PTP mode. Logically, the results obtained for a joram 3.5 durable subscriber stand between the results of a 3.4 subscriber and a 3.5 non durable subscriber.

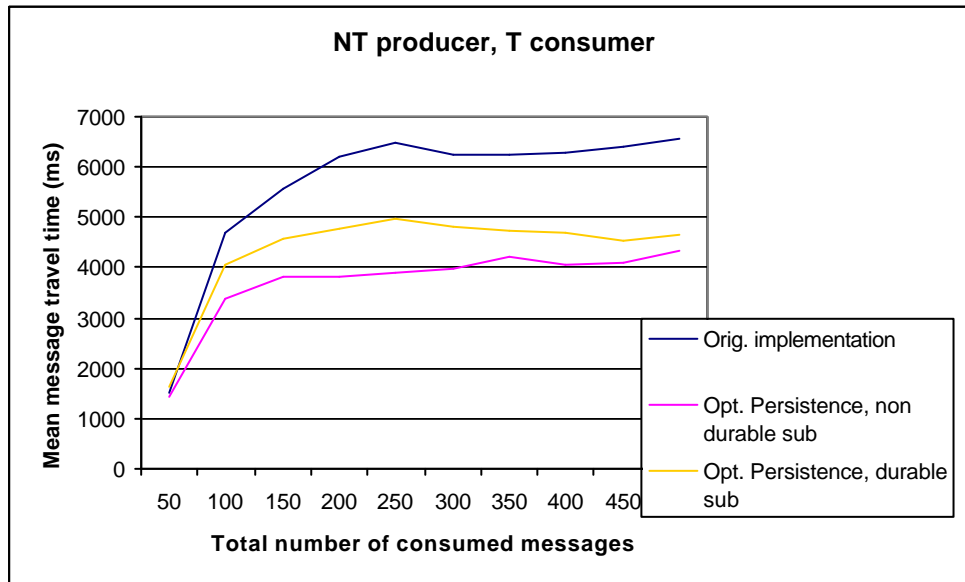
### Case 1: non transacted producer and consumer



Case 2 : transacted producer, non transacted consumer



Case 3 : non transacted producer, transacted consumer



Case 4: transacted producer and consumer

