

# A Transparent Light-Weight Group Service

Luís Rodrigues and Katherine Guo

(INESC)

(Cornell University)

joint work with

A. Sargento (INESC), R. van Renesse (Cornell),

B. Glade (ISIS), P. Veríssimo (INESC),

K. Birman (Cornell)

# Introduction

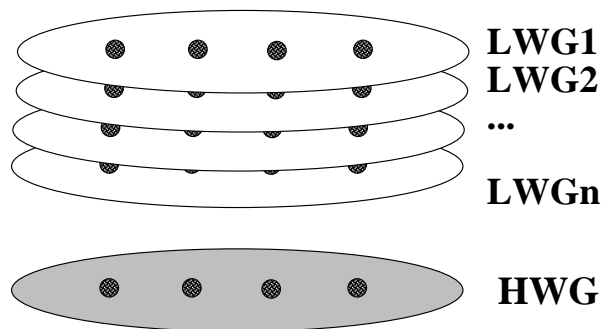
Virtual Synchrony:

- delivers group membership info using *Views*
- orders messages with *view* changes

Implementation requirement:

- failure detector
- agreement and ordering protocols

Optimization: when several groups have a large percentage of common members



# Outline

- Design Overview
- LWG Service Interface
- Protocols
  - The Flush Protocol
  - Failure Handling Protocol
  - Message Passing Protocol
- Performance Results
- Conclusions and Future Work

# Design Overview

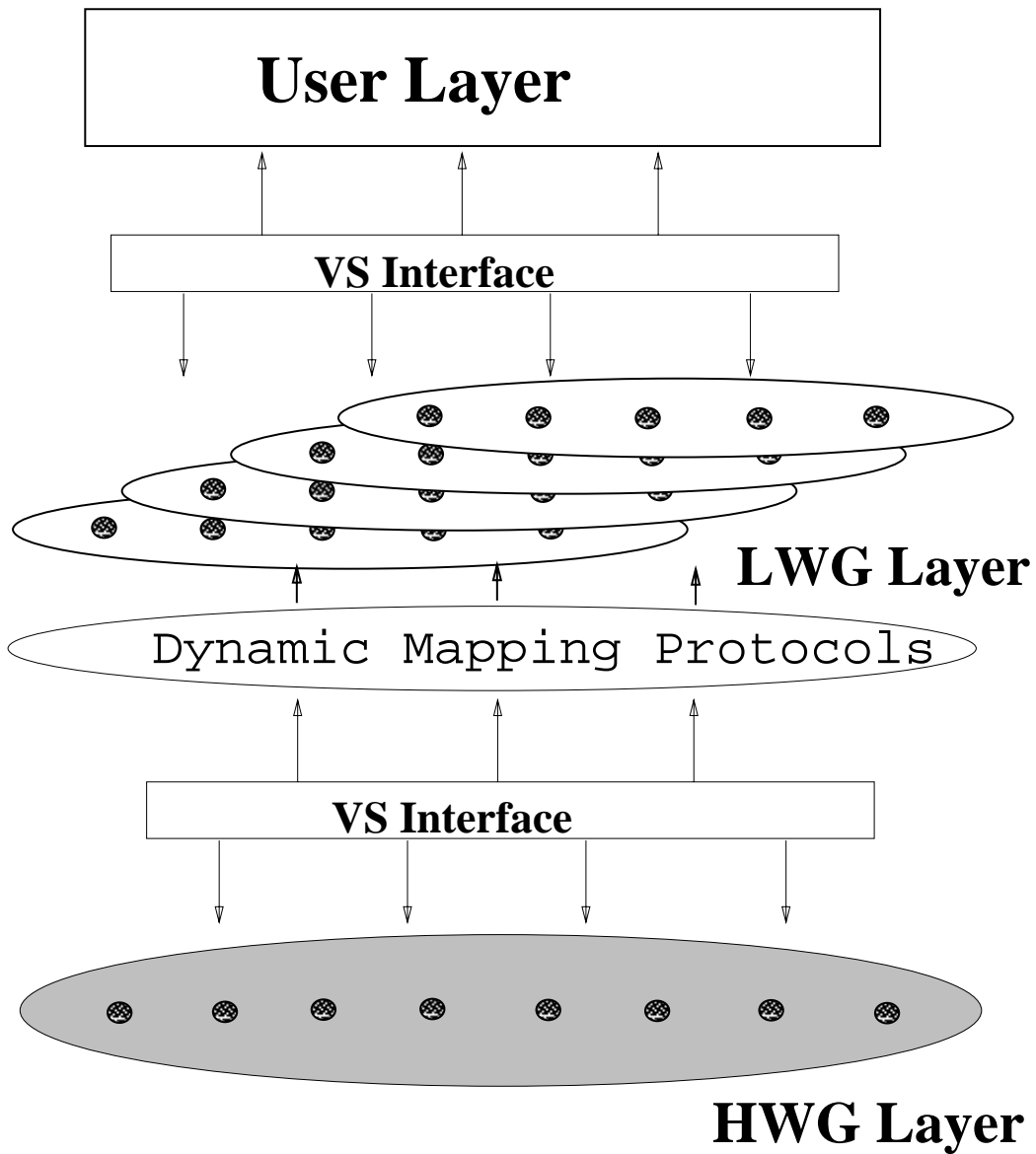
Main Tasks of the LWG Service:

1. preserve the VS interface of the HWGs to LWG users.

- keep a pool of HWGs.
- establish mappings between LWGs and HWGs.
- switch a LWG from  $HWG_{old}$  to  $HWG_{new}$  when  $HWG_{old}$  is not appropriate anymore.

2. define the mapping and switching policies [GR96].

# LWG Service Interface



# Protocols

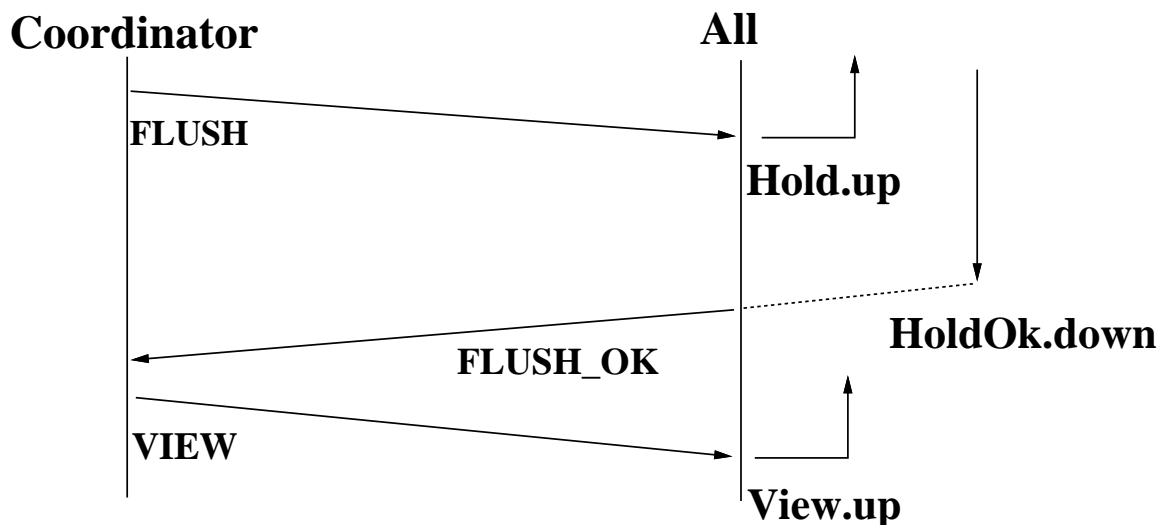
run on top of a VS communication service  
(Isis, Horus and NavTech)

- Join a group
- Leave a group
- Reconfigure a group after process crashes
- Multicast messages in a group
- Switch a LWG from one HWG to another HWG

# The Flush Protocol

Two places to use the protocol:

- to change LWG membership
- to switch to another HWG



**Hold.up:** ask appl to stop sending messages.

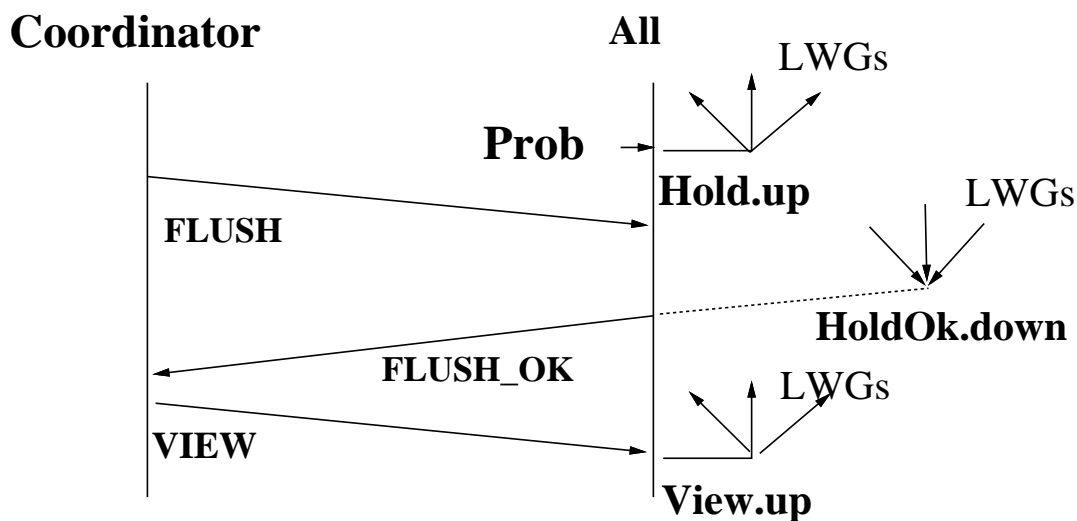
**HoldOK.down:** appl stops sending messages.

# Failure Handling Protocol

step 1. When failure is detected at a member, **Hold.up** is multiplexed to all LWGs on the HWG.

Coordinator of HWG detects failure and sends FLUSH in HWG.

step 2. LWG Service waits for an ack (**HoldOK.down**) from every LWG then acknowledges the **Hold.up** with **HoldOk.down**



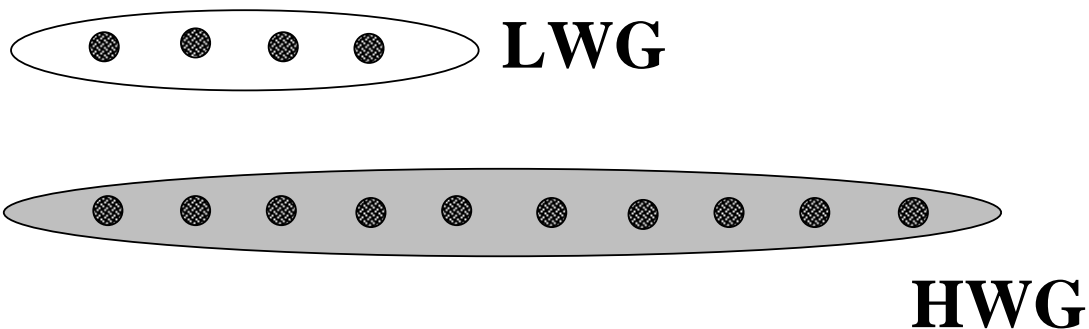
**Hold.up**: ask appl to stop sending messages.

**HoldOK.down**: appl stops sending messages.

# Message Passing Protocol

**send:** LWG encapsulates the LWG data in  
<DATA, *lwgid*, data>

**receive:** examines *lwgid* and forwards the  
message to the LWG.

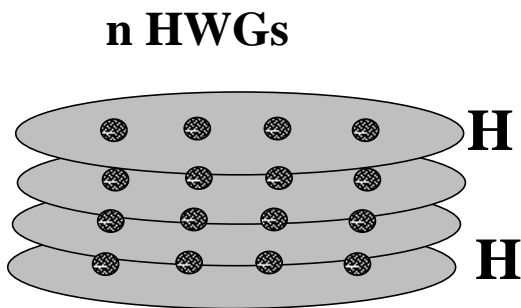


[GR96]

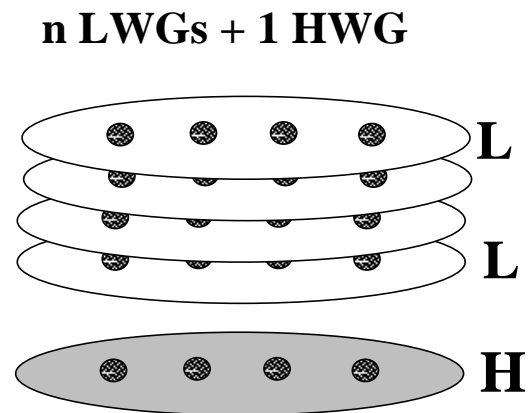
## Performance Test

- $n$  identical 4-member groups on top of 4 machines.
- each machine has one process.

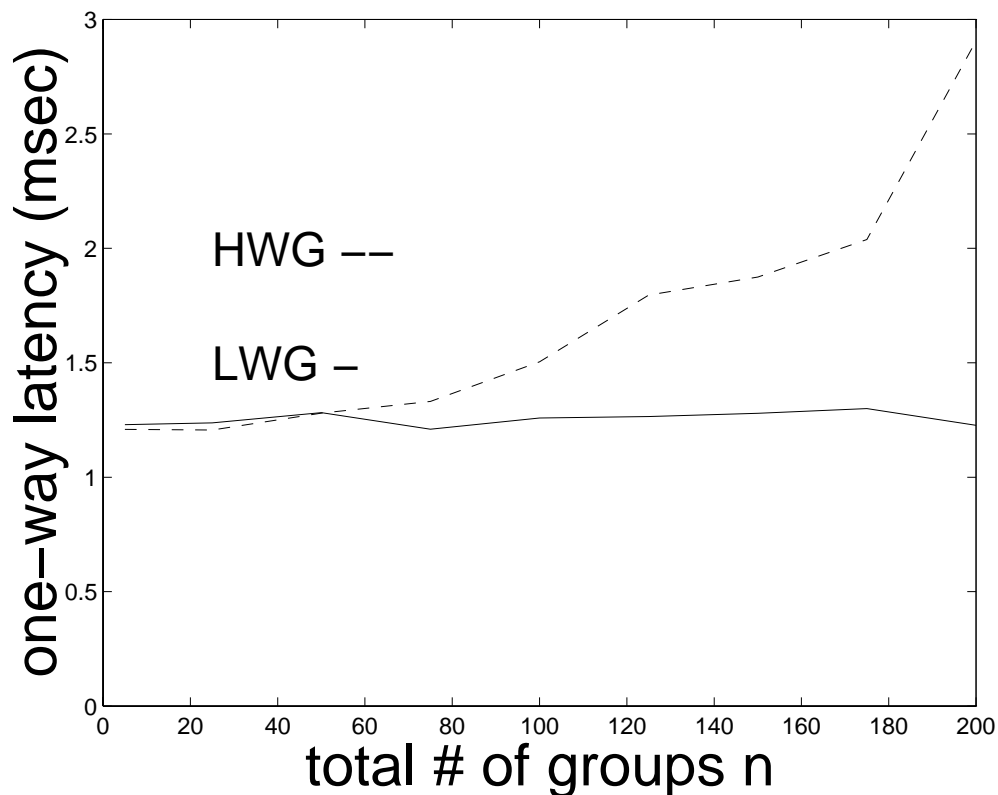
HWG test



LWG test



## Performance (data transfer)



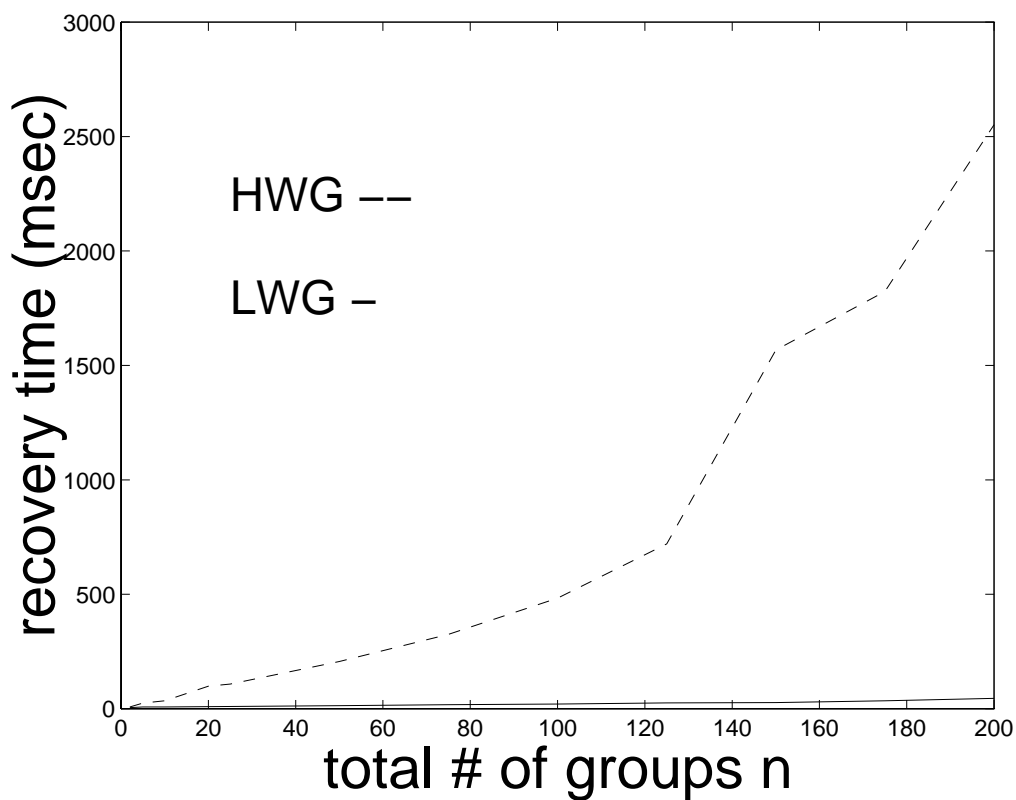
explanation: each member multicasts 1 "status" message every 2 seconds.

$n$  HWGs on each process  $\rightarrow 3n/2$  input "status" messages per second (4-member groups)

**bottleneck:** receiver out of buffer space  $\rightarrow$  retrans

## Performance (failure recovery)

test: a given process, member of  $n$  identical 4-member groups, crashes and forces these  $n$  groups to reconfigure.



$n$  HWGs

$n$  LWGs

$n$  HWG flushes in parallel

1 HWG flush

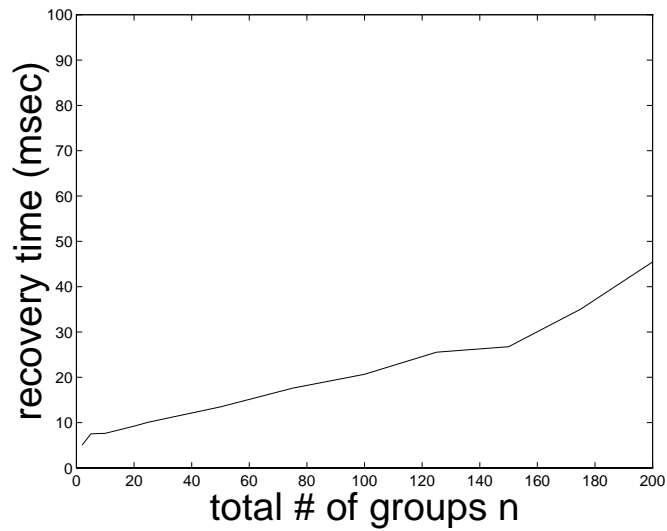
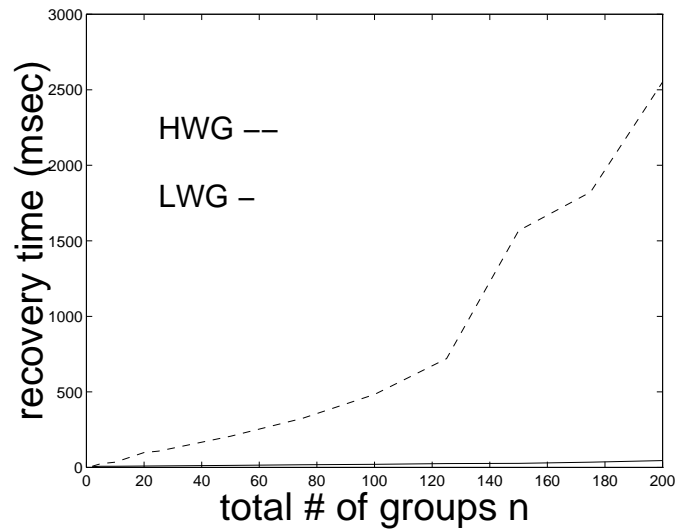
## Conclusions and Future Work

LWG Service:

- offers VS semantics transparently
- promotes resource sharing
- offers performance advantages

Future work: dynamically modify the mappings using switching heuristics [GR96].

# Performance (failure recovery)



$n$  HWGs

$n$  LWGs

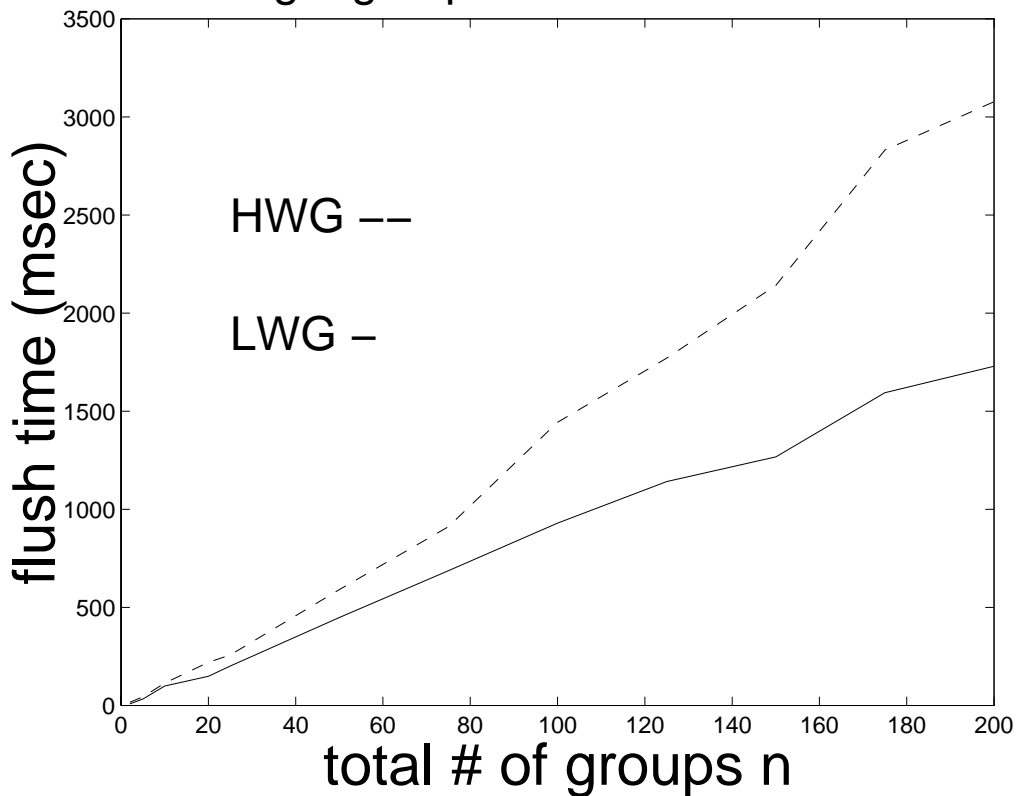
$n$  HWG flushes in parallel

1 HWG flush

## Performance (join $n$ groups)

test: join  $n$  groups as the 4th member.

Joining  $n$  groups as the fourth member.



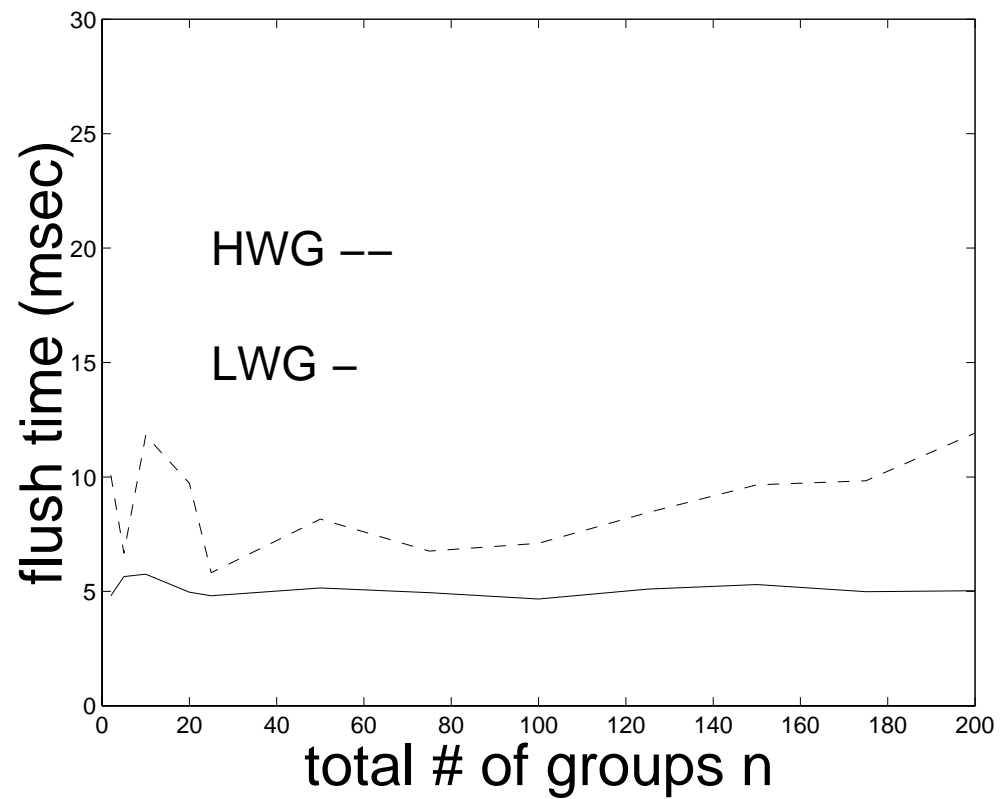
**JoinTime**( $p, n$ ): time for joining  $n$  groups of  $p - 1$  members.

**Flush**( $p$ ): time for each flush with the resulting group size  $p$ .

$$\text{JoinTime}_{\text{HWG}}(p, n) = \text{Flush}_{\text{HWG}}(p) \times n$$

$$\text{JoinTime}_{\text{LWG}}(p, n) = \text{Flush}_{\text{HWG}}(p) + \text{Flush}_{\text{LWG}}(p) \times n$$

## Performance (leave one group)



## Related Work

- Delta-4 LWG: static mapping between LWGs and HWGs
- Isis LWG: dynamic mapping

Not transparent: original HWG interface is not preserved.