Worker Nodes On Demand

Running batch jobs in customized environments

Davide Salomoni

INFN-CNAF

(davide.salomoni@cnaf.infn.it)

This talk

- Problem statement
  - Related to customer support
  - Related to general batch farm issues
- The conflicting wish lists
- Using virtualization to tackle the problem
  - The architecture developed at CNAF
- Evaluation / evolution

Problem statement: customer support issues

- Supporting multiple experiments normally means one has to deal with diverse customer requirements
  - Operating System
    - "I want SLC3" - "Hey no, my application [only runs | is only certified] on Ubuntu 8.04" - "What? Forget it! I need afs and SL5"
    - "Would you please upgrade all your worker nodes to a 64-bit OS *by this week*?"
  - Applications
    - "I *absolutely* need you to install application X.Y version Z on all your nodes"
    - "Please don't change *that* system library!" ("and don't you dare to upgrade the kernel!")
  - Intra-VO requirements may also apply
    - Different set of users belonging to the same VO may raise different requirements
  - The INFN Tier-1 currently supports ~20 Virtual Organizations

Problem statement: customer support issues

- Supporting multiple experiments normally means one has to deal with diverse customer requirements
  - Ope
    - 04" -
    -
  - App
    -
    - rnel!")
  - Intra
    -
  - The INFN Tier-1 currently supports ~20 Virtual Organizations

Problem statement: batch farm issues

- Consider some effects of running multiple jobs on a single, shared system
  - How to prevent a given job to steal (willingly or not) more than a fair share of resources? Think memory leak, for instance.
  - What about processes losing the connection with their parent, escaping batch system checks, and becoming children of *init*?
  - How about security exploits damaging other users running on the node?
  - What if finished jobs left over [a significant amount of] data? (local storage shortage)
  - Out Of Memory (OOM) killer on systems with more than 8GB RAM *and* a 32-bit kernel (cf. in general, exhaustion of the low memory address space)
- Effects amplified the more shared a given "Worker Node" becomes → typically, the more cores/job slots a WN has
  - Example: a common 2x quad-core at the INFN Tier-1 has 10 job slots (25% overbooking)

Problem statement: batch farm issues

- Consider some effec on a single, shared system
  - How to prevent a given j a fair share of resources? Think memory leak, for instanc
  - What about processes lo escaping batch system checks, and becoming children o
  - How about security explo the node?
  - What if finished jobs left ocal storage shortage)
  - Out Of Memory (OOM) k RAM *and* a 32-bit kernel (cf. in general, exhaustion of th
- Effects amplified the rker Node" becomes → typically, the more co
  - Example: a common 2x ob slots (25% overbooking)

The conflicting wish lists

- Customer
  - I am the [only | most important | most powerful] customer of your site, so listen to me
- Site
  - Optimize resource usage
    - Avoid static allocations; try to avoid wasting CPU cycles
  - Don't [buy, set-up] a separate, dedicated infrastructure to fix requirements / problems
    - Minimize additional costs
  - Know who's running what and where
  - Do not change established work flows
  - Maintain full control of the site

Tentative answer: run jobs in dedicated environments

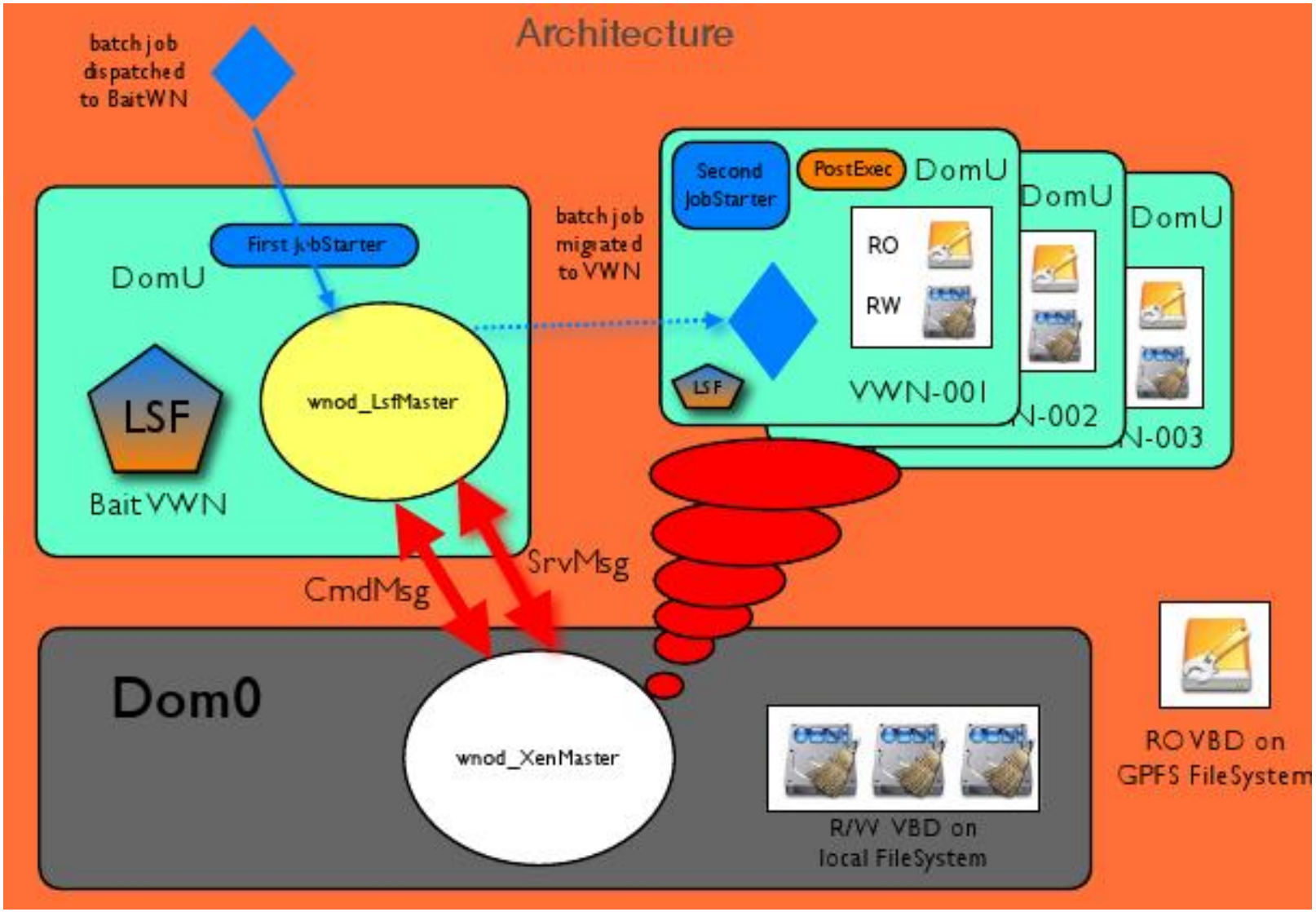
- If one could isolate jobs so that they run in dedicated environments, then a number of the aforementioned issues would be solved
  - And if one could also customize the dedicated environment...
  - And have this working dynamically (i.e. without too many static assumptions)...
- Dedicated environment → Virtualization!
  - But how? (“the devil is in the details”)

Worker nodes on virtual machines?

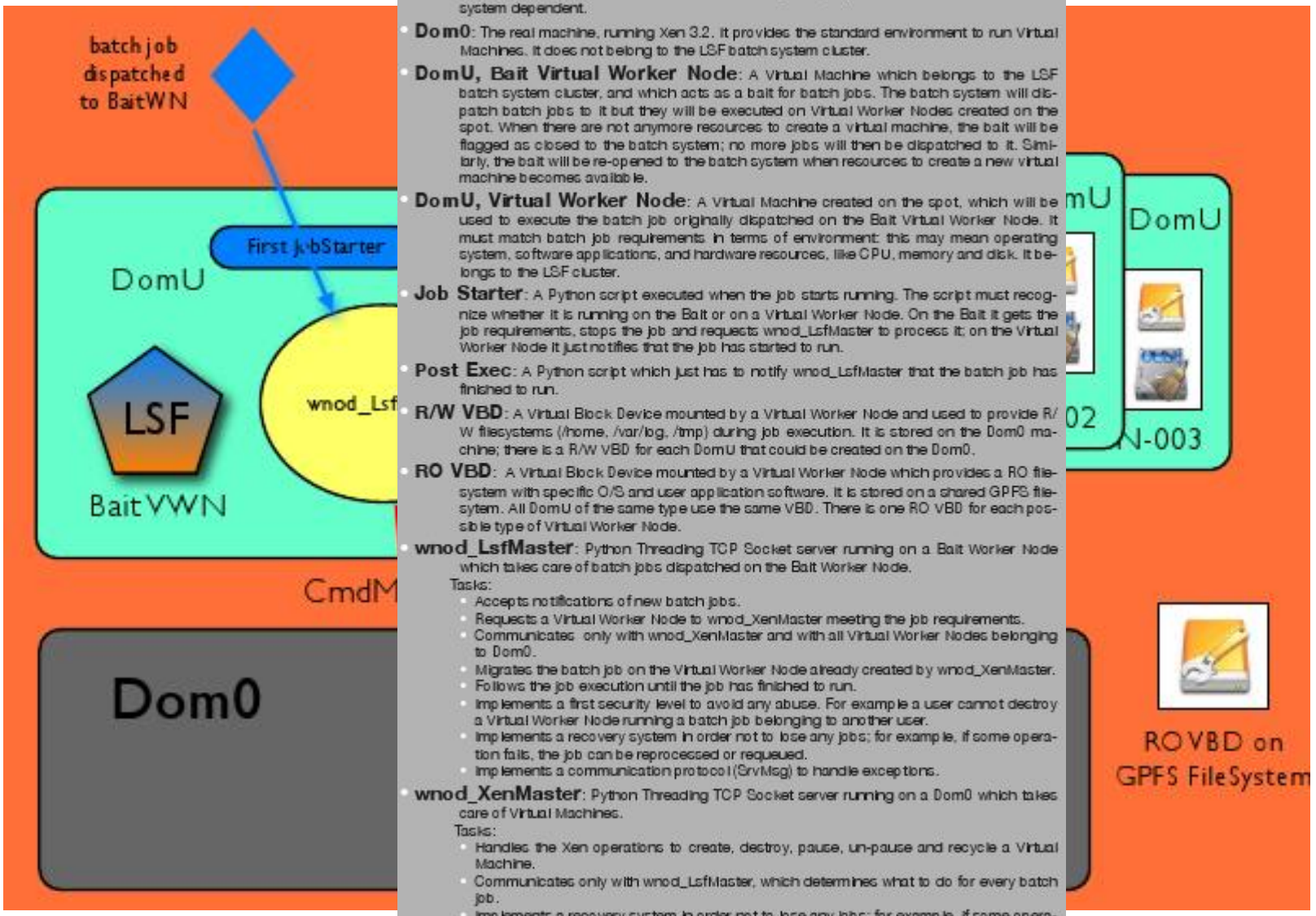
- Before considering virtualization as a viable possibility, the following questions (at least) should be answered:
  - Is it stable? Scalable?
  - How much performance penalty?
  - Where would you put the VMs?
    - How efficient is that? (take e.g. startup time, network traffic, possible caching)
  - What about integration? E.g., with the...
    - ... LRMS (the batch system)
      - And its licensing scheme (for those of us running commercial LRMS, that is)
    - ... grid middleware
    - ... monitoring, accounting, installation subsystems
    - ... upgrade procedures

General Architecture

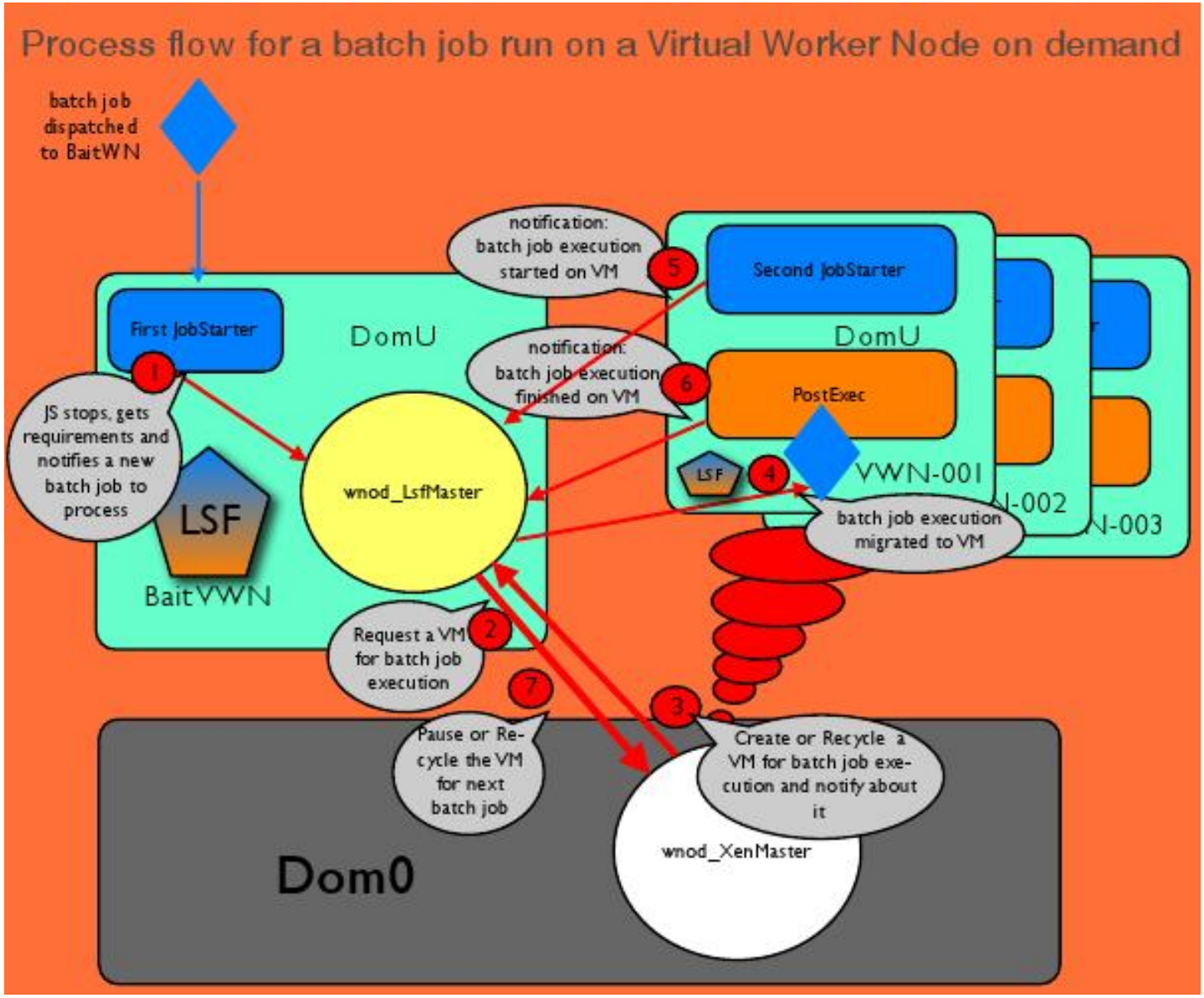
- Xen-based
  - Dom0 is strictly LRMS-unaware
  - On each physical hardware hosting VMs, there is a special DomU acting as *bait* (a job attractor)
    - Nowhere in the system there is either a single point of failure (besides those possibly existing in a solution without VMs, that is), or a dispatching engine competing with the one (hopefully highly optimized and tuned) of the LRMS
  - Other DomU on the physical hardware are the virtual worker nodes
    - *Created on the spot* when a job arrives, or *re-used* if caching is enabled
  - The VM images are divided in two parts: a strictly R/O one on a shared file system, and a R/W on the local system
    - This currently limits the solution to Unix-like O/S
  - The LRMS licensing requirements increase by 1 license per physical system (w/o overbooking) – due to the bait DomU


Architecture
batch job dispatched to BaitWN
First JobStarter
DomU
LSF
BaitVWN
wnod_LsfMaster
batch job migrated to VWN
Second JobStarter
PostExec
DomU
RO
RW
LSF
VWN-001
DomU
N-002
DomU
N-003
CmdMsg
SrvMsg
Dom0
wnod_XenMaster
R/W VBD on local FileSystem
RO VBD on GPFS FileSystem

Details of the components



- **Batch system:** Platform LSF in our case, although the proposed architecture is not batch system dependent.
- **Dom0**: The real machine, running Xen 3.2. It provides the standard environment to run Virtual Machines. It does not belong to the LSF batch system cluster.
- **DomU, Bait Virtual Worker Node**: A Virtual Machine which belongs to the LSF batch system cluster, and which acts as a bait for batch jobs. The batch system will dispatch batch jobs to it but they will be executed on Virtual Worker Nodes created on the spot. When there are not anymore resources to create a virtual machine, the bait will be flagged as closed to the batch system; no more jobs will then be dispatched to it. Similarly, the bait will be re-opened to the batch system when resources to create a new virtual machine becomes available.
- **DomU, Virtual Worker Node**: A Virtual Machine created on the spot, which will be used to execute the batch job originally dispatched on the Bait Virtual Worker Node. It must match batch job requirements in terms of environment: this may mean operating system, software applications, and hardware resources, like CPU, memory and disk. It belongs to the LSF cluster.
- **Job Starter**: A Python script executed when the job starts running. The script must recognize whether it is running on the Bait or on a Virtual Worker Node. On the Bait it gets the job requirements, stops the job and requests wnod_LsfMaster to process it; on the Virtual Worker Node it just notifies that the job has started to run.
- **Post Exec**: A Python script which just has to notify wnod_LsfMaster that the batch job has finished to run.
- **R/W VBD**: A Virtual Block Device mounted by a Virtual Worker Node and used to provide R/W filesystems (/home, /var/log, /tmp) during job execution. It is stored on the Dom0 machine; there is a R/W VBD for each DomU that could be created on the Dom0.
- **RO VBD**: A Virtual Block Device mounted by a Virtual Worker Node which provides a RO filesystem with specific O/S and user application software. It is stored on a shared GPFS filesytem. All DomU of the same type use the same VBD. There is one RO VBD for each possible type of Virtual Worker Node.
- **wnod_LsfMaster**: Python Threading TCP Socket server running on a Bait Worker Node which takes care of batch jobs dispatched on the Bait Worker Node.
  Tasks:
  - Accepts notifications of new batch jobs.
  - Requests a Virtual Worker Node to wnod_XenMaster meeting the job requirements.
  - Communicates only with wnod_XenMaster and with all Virtual Worker Nodes belonging to Dom0.
  - Migrates the batch job on the Virtual Worker Node already created by wnod_XenMaster.
  - Follows the job execution until the job has finished to run.
  - Implements a first security level to avoid any abuse. For example a user cannot destroy a Virtual Worker Node running a batch job belonging to another user.
  - Implements a recovery system in order not to lose any jobs; for example, if some operation fails, the job can be reprocessed or requeued.
  - Implements a communication protocol (SrvMsg) to handle exceptions.
- **wnod_XenMaster**: Python Threading TCP Socket server running on a Dom0 which takes care of Virtual Machines.
  Tasks:
  - Handles the Xen operations to create, destroy, pause, un-pause and recycle a Virtual Machine.
  - Communicates only with wnod_LsfMaster, which determines what to do for every batch job.
  - Implements a recovery system in order not to lose any jobs; for example, if some operation fails, it sends a SrvMsg to wnod_LsfMaster to request a reprocessing or re-queuing of the batch job.
  - Implements a communication protocol (SrvMsg) to handle exceptions.


Process flow for a batch job run on a Virtual Worker Node on demand
batch job dispatched to BaitWN
First JobStarter
DomU
JS stops, gets requirements and notifies a new batch job to process
LSF
BaitVWN
wnod_LsfMaster
notification: batch job execution started on VM
notification: batch job execution finished on VM
Second JobStarter
DomU
PostExec
LSF
VWN-001
batch job execution migrated to VM
Request a VM for batch job execution
Pause or Recycle the VM for next batch job
Create or Recycle a VM for batch job execution and notify about it
wnod_XenMaster
Dom0

Current status

- The system has been tested with tens of VMs, without significant architectural issues
  - Seamless integration with the existing installation, monitoring, and accounting subsystems, with both local- and grid- type of job submission, and with the existing shared file system
    - Accessing the shared file system in R/O mode greatly reduces load, while still providing consistency
  - We'd like to extend the testbed to hundreds of VMs by this summer
  - Some work needs to be done in advance for the pre-packaging of the VMs (separation of the R/O and R/W parts)
    - The procedures to provide these images should be clearly defined between the Tier-1 and the VOs.

Evolution

- Distributing caches to further enhance efficiency?
- Integration with the Glue schema to publish information of the virtual resources provided by the system?
- Ports to other LRMS?
  - While the system described here is not batch system dependent, the current implementation has been written to work with Platform LSF
- To what extent is this cognate to “cloud computing”? (see e.g. Reservoir, http://www.reservoir-fp7.eu/)
- There are obviously other solutions, possibly covering different application scenarios
  - See e.g. the work of L.Servoli et al., INFN-PG
  - It would be very interesting to compare the alternatives

Question Time