Condor

A Hunter of Idle Workstations

Timur Chabuk

Big Idea

- workstations have:
  - more power than is usually needed
    - only 30% of capacity is used
    - many idle workstations, even at peak times
  - not enough power as what is sometimes needed

- Condor: schedules long running jobs at idle workstations

System Design Principles

- placement of jobs should be transparent to user
- failed jobs should be automatically restarted
- sharing means you get access
- system should consume little resources

Scheduler Options

Centralized Scheduler: gathers information and then schedules jobs

+ efficiently decide which job is granted
+ knows everything
- must be secure from users
- not easily extendable
- single point of failure

Distributed Scheduler: workstations cooperate

- message passing is less efficient
+ no single point of failure
+ scales better

Condor Scheduler

- Central coordinator
  - tracks availability of resources and jobs waiting
    - polls every 2 minutes
  - allocates capacity to workstations
- Workstation
  - keeps track of its own jobs and their priority, schedules accordingly
  - checks between polls to see if it has capacity
  - if job is running, checks every 1/2 minute to see if user has returned

Remote Unix Facility

- When invoked, a shadow process starts
  - runs locally
  - surrogate of process running remotely
  - system calls on remote machine invoke library routines that communicate with shadow
- Checkpointing is supported
  - important when user returns
  - saves state of program, allows for restarting
  - Nest, V-Kernel, Process Server do not

Fair Access: UpDown

- goals:
  - provide heavy users with steady access
  - provide fair access to light users
- workstations have “priority” rating
  - capacity granted -> decrease priority
  - job request denied -> increase priority
- coordinator checks every 2 minutes
  - can preempt low-priority jobs with high-priority jobs

Performance

- very low wait ratio for light users
- 10 ms for system calls
  - 20 times slower than direct Unix calls
- leverage (remote capacity consumed / local capacity consumed)
  - average of 1300 (1 minute local for 22 hours)
  - 600 for jobs of less than 2 hours
- local scheduler uses < 1% of CPU (scales)
- coordinator uses < 1% of CPU (doesn’t scale)

Condor and the Grid

Philosophy of Flexibility

- Let communities grow naturally
  - People can figure things out. Let them.
- Leave Owner in Control
  - Attract maximum number of users.
- Plan without being picky
  - Hardware will fail, things will go wrong, plan for it.
- Lend and Borrow
  - Share benefit of experience, learn from other disciplines.
- Research
  - Look at what's been done.

Condor Today

- 5 main tasks:
  - Research in Distributed Computing
    - harnessing opportunistic and dedicated resources
    - job management for grid apps
    - fabric management for grid services
    - resource discovery, monitoring, management
    - problem solving environment
    - distributed I/O technology
  - Participation in Scientific Community
  - Engineering of Complex Software
  - Maintenance of Production Environments
  - Education of Students

Condor

- High throughput computing
  - Fault tolerant computing, for long durations
- Opportunistic computing
  - Use resources whenever available

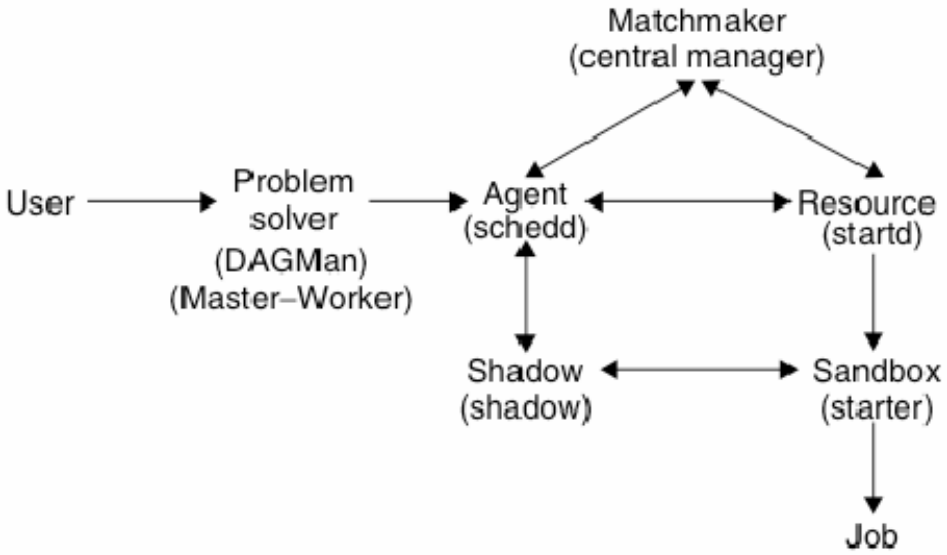
Condor-G

- marriage of Globus and Condor
- Globus provides mechanisms for
  - secure interdomain communication
  - standardized access to remote batch systems
- Condor provides mechanisms for
  - job submission
  - job allocation
  - error recovery
  - friendly execution environment

Basic Kernel

Agent

- may be more than one per machine
- receives jobs from users
- advertises itself to Matchmaker
- finds resources to run jobs on (via Matchmaker)
- enforces user's policies on what resources to use



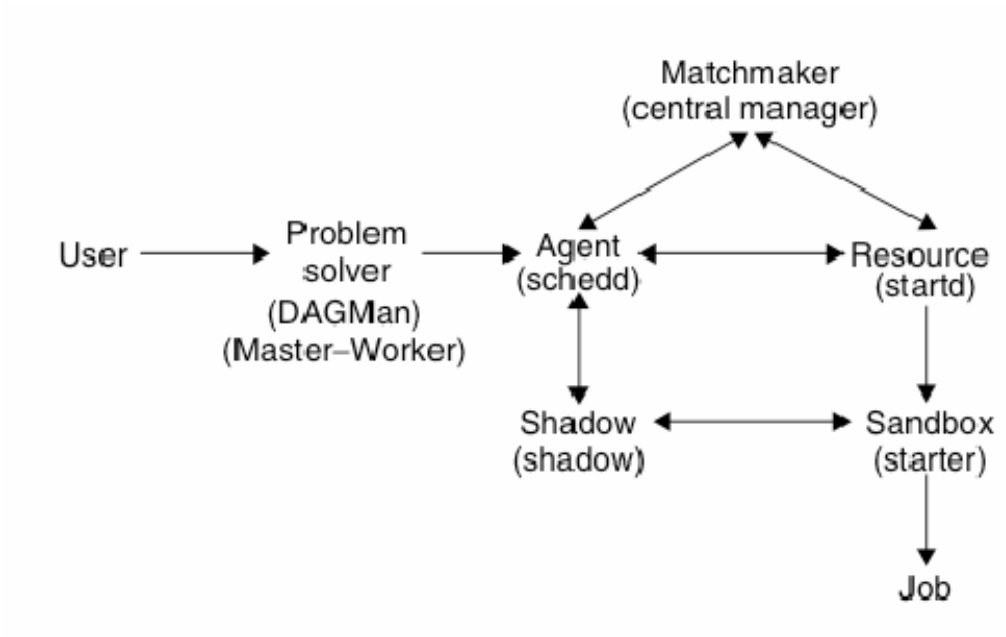
Basic Kernel

Resource

- advertises itself to Matchmaker
- enforces resource's policies on what users to trust

Matchmaker

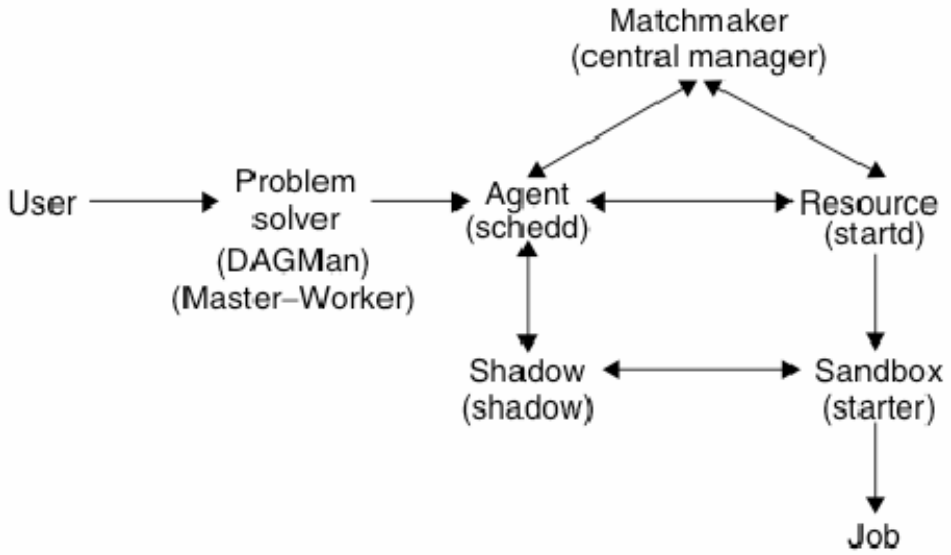
- introduces compatible agents and resources
- enforces community policies



Basic Kernel

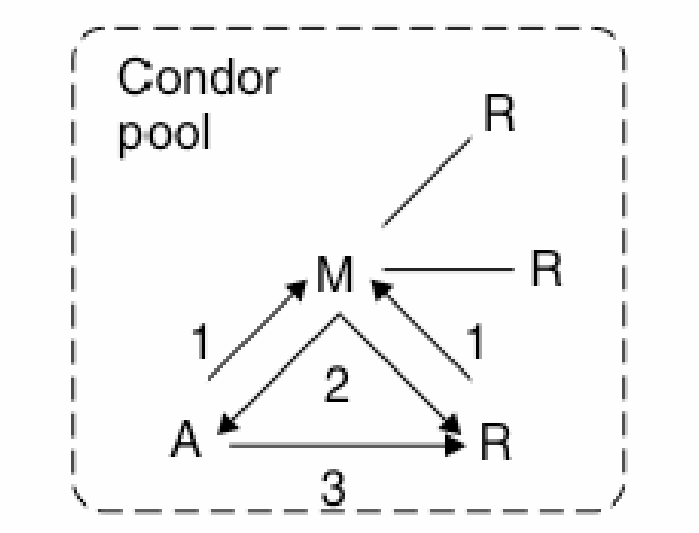
Once introduced…

- Agent contacts resource and validates match
- Shadow is launched at Agent
  - provides details to run job
- Sandbox is launched at Resource
  - provides safe execution environment



Computing Communities: Condor Pools

- community is defined by well-known Matchmaker
- Matchmaker enforces community policies
- issues:
  - users may only participate in one community
  - cannot share across organizational boundaries



Computing Communities: Gateway Flocking

Gateways (one per pool):

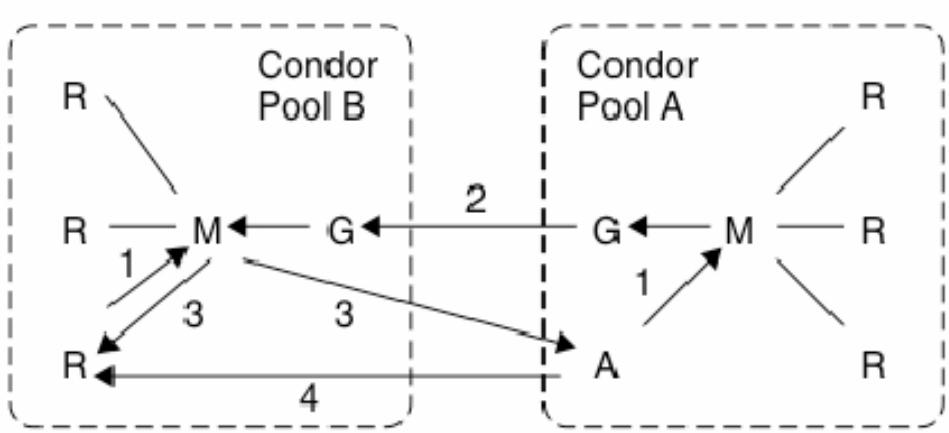
- detects resources/agents from pool's Matchmaker
- communicates to peer
- peer then communicates to it's pool's Matchmaker

Benefits

- transparent to user
- incremental system growth

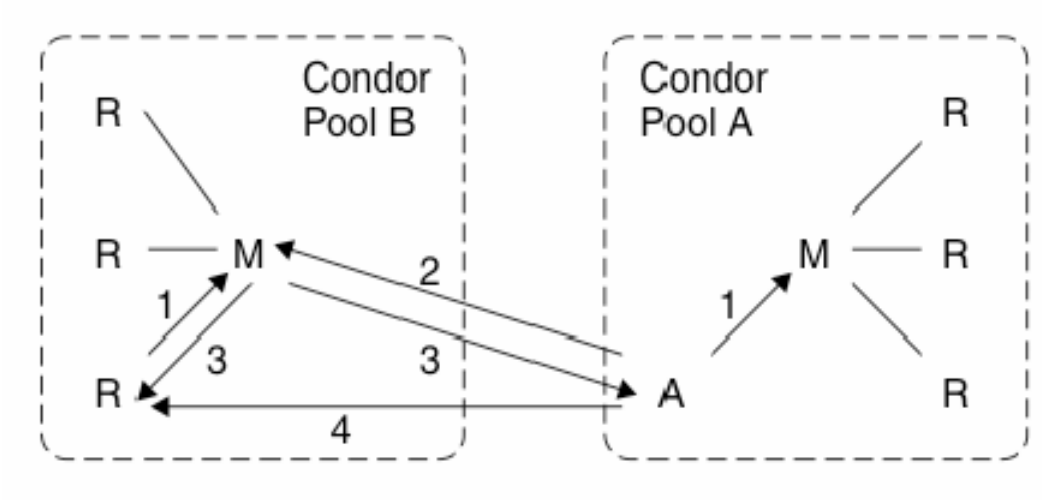
Drawbacks

- only allows sharing at organizational level
- user cannot be part of more than one community



Computing Communities: Direct Flocking

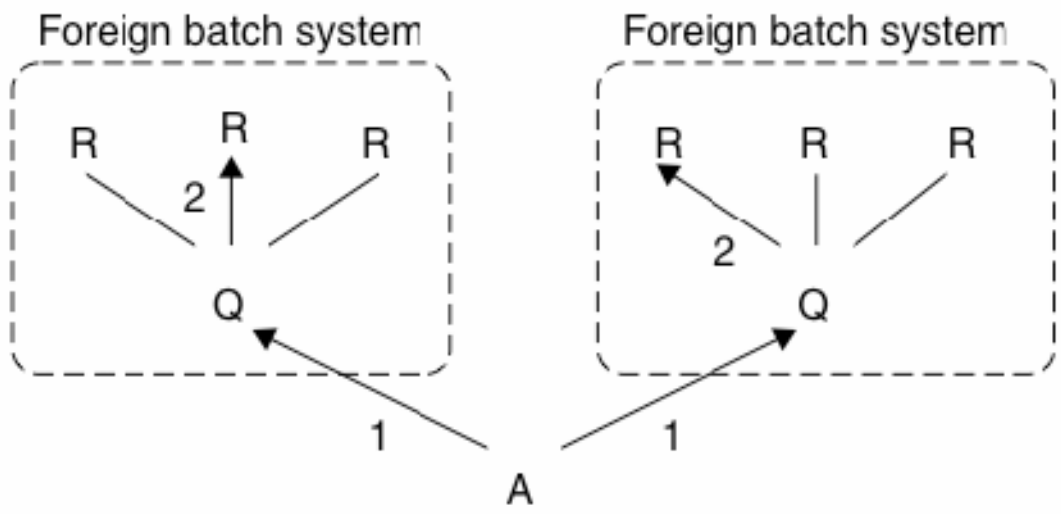
- Agents report to multiple communities' Matchmakers
  - + agreement between individual and organization only
  - - not transparent to user



- Gateway did not last
  - – too complex

Computing Communities: GRAM and Condor-G

- GRAM: abstraction for remote process queuing and execution
- Condor-G: implementation of Condor that “speaks GRAM”



Computing Communities: Condor-G

Benefits:

- user can reach any sort of batch system

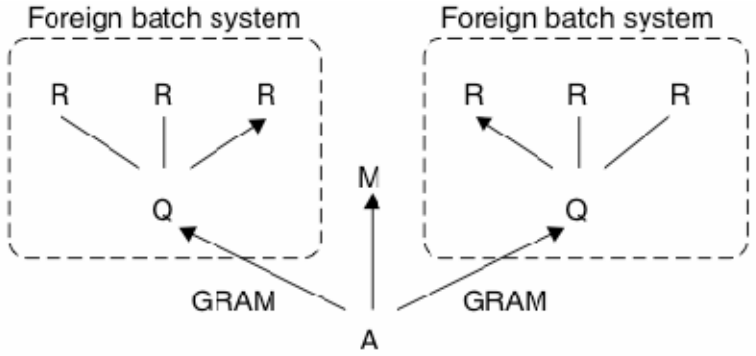
Drawbacks:

- couples resource allocation and job execution
- agent will over or undersubscribe
- checkpointing not supported

Condor-G: “Gliding In”

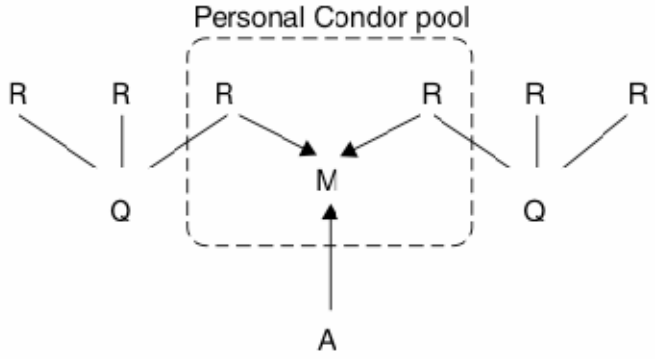
Step one:

User submits Condor daemons as batch jobs in foreign systems



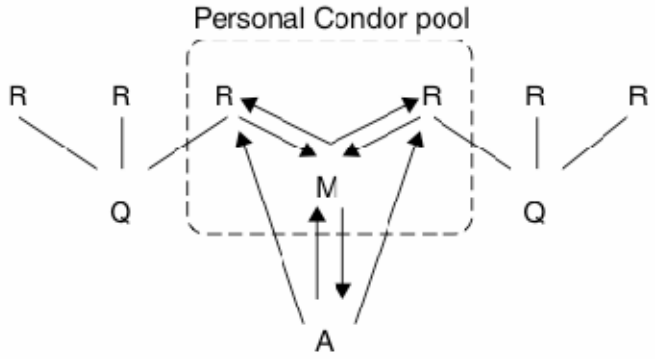
Step two:

Submitted daemons form an *adhoc* personal Condor pool



Step three:

User runs jobs on personal Condor pool



Computing Communities: I/O

- resources group by being “nearby”
  - network latency, system throughput, etc.,
- execution domain
  - all resources identify to a checkpoint server
  - agent uses physical information to make placement/migration decisions

Planning and Scheduling

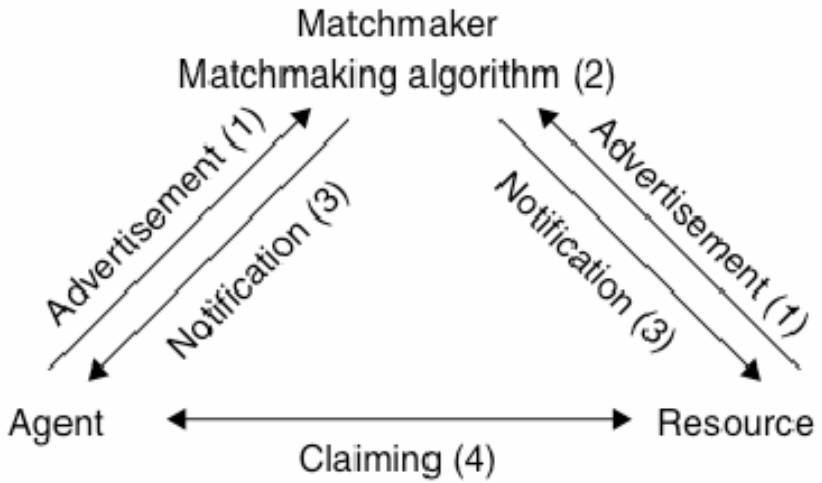
- Planning: acquisition of resource by user
  - what and where
- Scheduling: management of resource by owner
  - who and when

- Matchmaker bridges the gap
  - allows them to work together
  - respects their independence

Planning and Scheduling

Four Steps

- Classified Advertisements
  - attribute name - value pairs
  - user defined requires 3-valued logic
  - Requirements (constraints) and Rank (preference)
- Matchmaker
- Notification
- Claiming
  - separation allows for validation at current time



Combinations of Planning and Scheduling

- Planning around a Schedule:
  - remote scheduler publishes information about its timetable in the ClassAds
  - Matchmaker can make better decisions
- Scheduling with a Plan
  - similar to “gliding-in”?

Matchmaking in Practice

- standalone software packages for manipulating ClassAds (Java and C++)
- extensions: gang matching, collections, set matching, indirect references
- stateless -> scales well without complex failure recovery

Problem Solvers

- higher-level structure built on top of Condor Agent
- uses Agent to reliably execute jobs
- is itself submitted as a normal Condor job
  - once started, can submit other jobs

Master-Worker

situations where…

- large portions of program are independent
- guided by intermediate results

components

- worklist:
  - outstanding work
- tracking
  - assigns remote workers tasks
- steering
  - examines results
  - modifies work list
  - gets workers from Condor

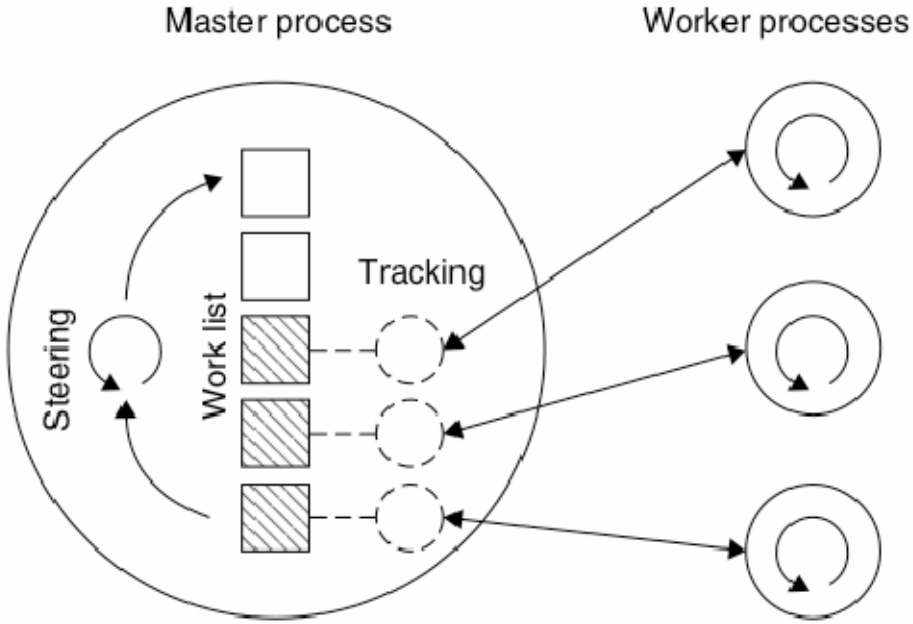
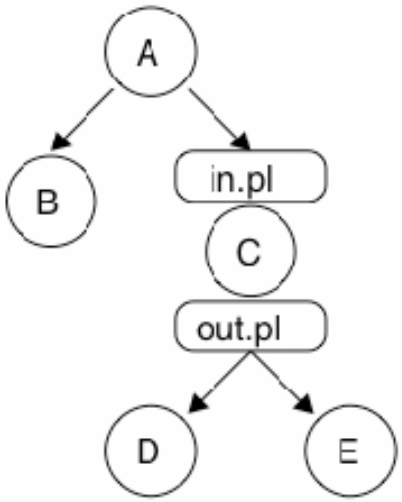


Figure 11.13 Structure of a Master–Worker program.

DAGMan

- similar to “makefile”
- dependencies, etc.,
- feedback on errors




```
JOB A a.condor
JOB B b.condor
JOB C c. condor
JOB D d.condor
JOB E e.condor
PARENT A CHILD B C
PARENT C CHILD D E
SCRIPT PRE C in.pl
SCRIPT POST C out.pl
RETRY C 3
```

Figure 11.14 A Directed Acyclic Graph.

Split Execution

- Submission and execution machines must cooperate
- Shadow
  - Represents the user to the system
  - Provides executable, arguments, etc.,
- Sandbox
  - Provides everything to run (sand)
  - Security for resource (box)
- Universe = a matched Shadow and Sandbox

Standard Universe

- reproduce user's home environment on remote site
- emulates majority of system calls
- checkpointing
- shadow:
  - represents user
  - provides job details
  - makes policy decisions
  - I/O over secure RPC

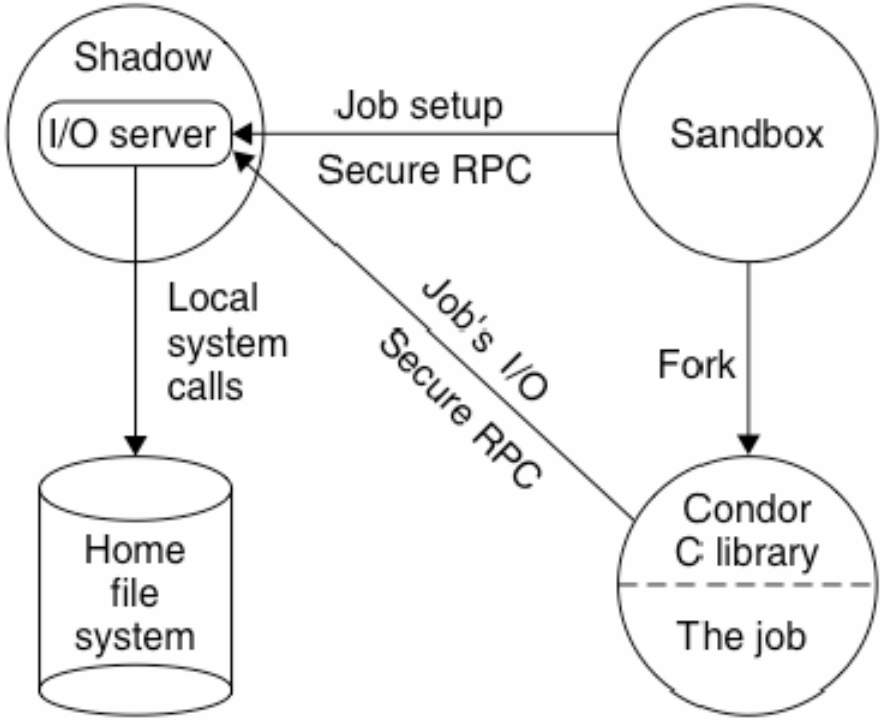


Figure 11.15 The standard universe.

Standard Universe: I/O

- user job is relinked with special library
  - interface of standard C library
  - converts all I/O ops into remote protocols
- shadow must remain in control (two-phase open)
  - sandbox and library must request resource from shadow
  - shadow provides information on how to access

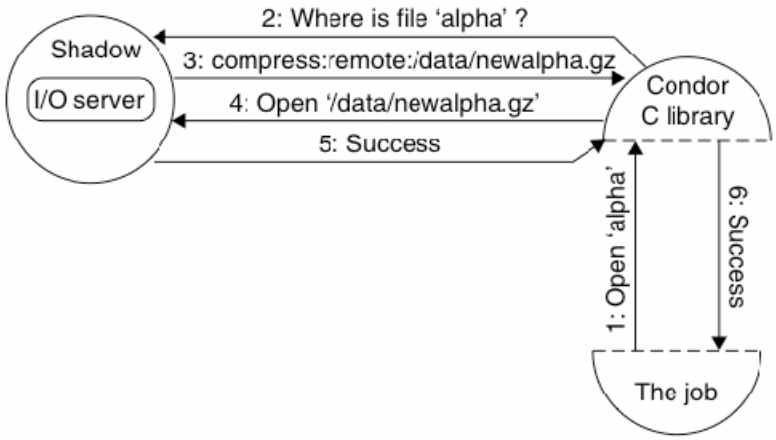


Figure 11.16 Two-phase open using the shadow.

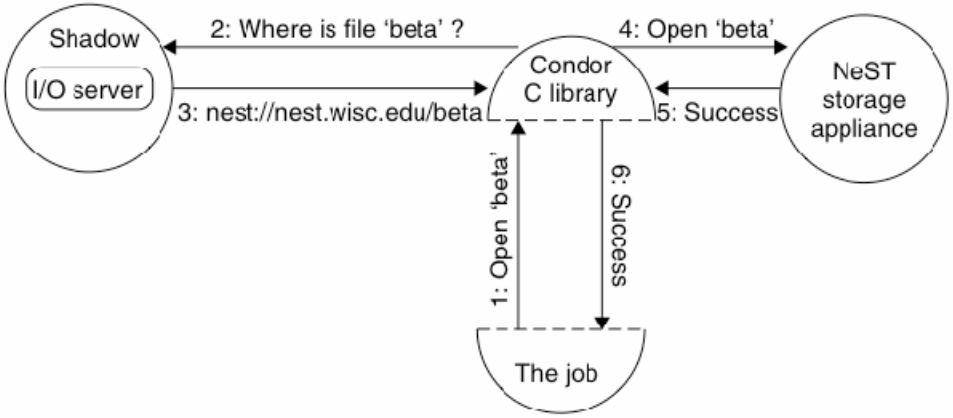


Figure 11.17 Two-phase open using a NeST.

Java Universe

- used to have to transfer entire JVM
- sandbox
  - places all runtime components in private dir
  - launches JVM

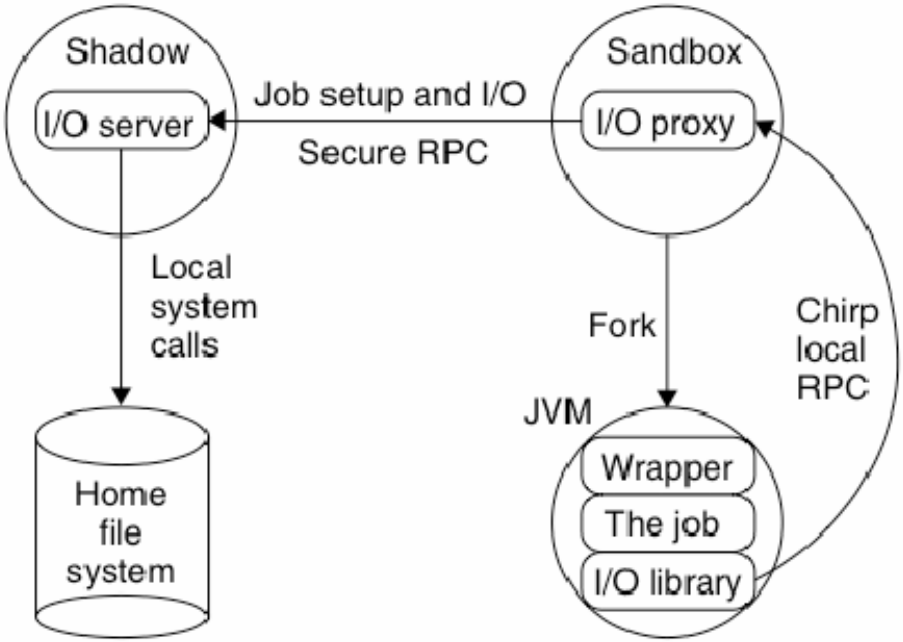


Figure 11.18 The Java universe.

Java Universe: I/O

- I/O library
  - implements standard interfaces
  - calls I/O proxy
- I/O proxy
  - executes via secure RPC
  - lets sandbox handle many issues (firewall, etc.,)

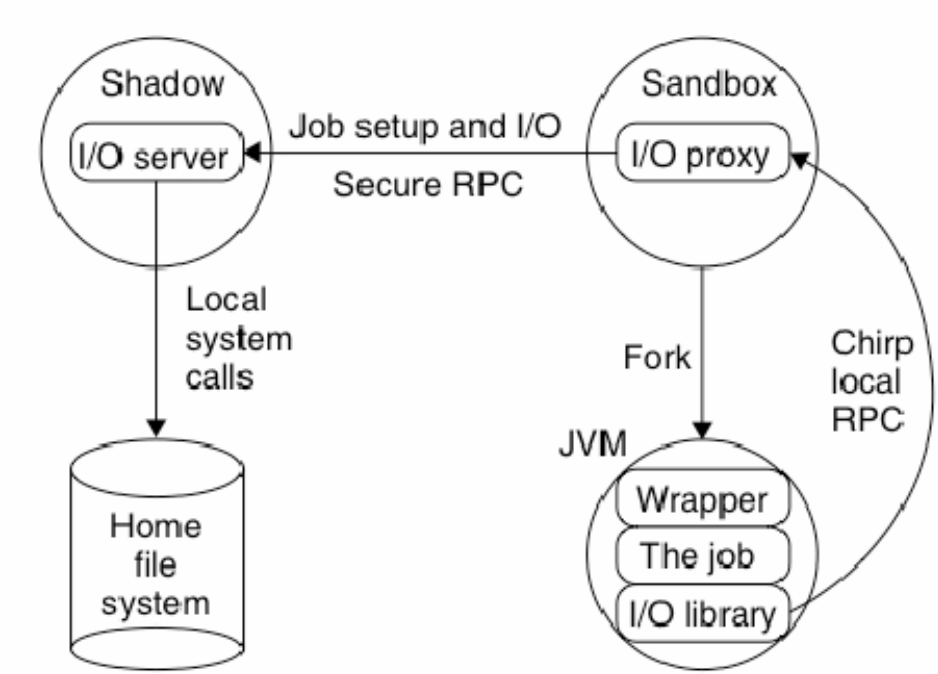


Figure 11.18 The Java universe.