Transparent Process Migration: Design Alternatives and the Sprite Implementation

F. Douglis and J. Ousterhout, UC Berkeley, SP&E, August 1991

Process migration

- Process migration – to move a process's execution site at any time from a source to a destination machine of the same architecture
- Alternatives like `rsh` lack transparency
  - The process doesn't run in the same environment as the parent process (cwd, environment variables, …)
- Eviction
  - You could only kill the process
- Performance
  - It's to slow which makes it impractical to short-lived processes
- Automatic selection of target machine
  - User has to pick where to run the process

Process migration in Sprite

- Sprite
  - Unix-like OS for a set of workstations and file servers in a LAN
  - High degree of network integration, e.g. common file server
  - Kernels work closely through RPC
- Key aspects to migration in Sprite
  - Idle hosts are plentiful – 66-87% idle in their case
  - Users ‘own’ their workstation – need quick eviction
  - Sprite uses kernel calls which may mean a harder environment than with message-passing system
  - It provides network support such as remote access to files and devices, single network-wide process IDs & efficient RPC
  - Sprite machines are typically diskless; files are access through file servers

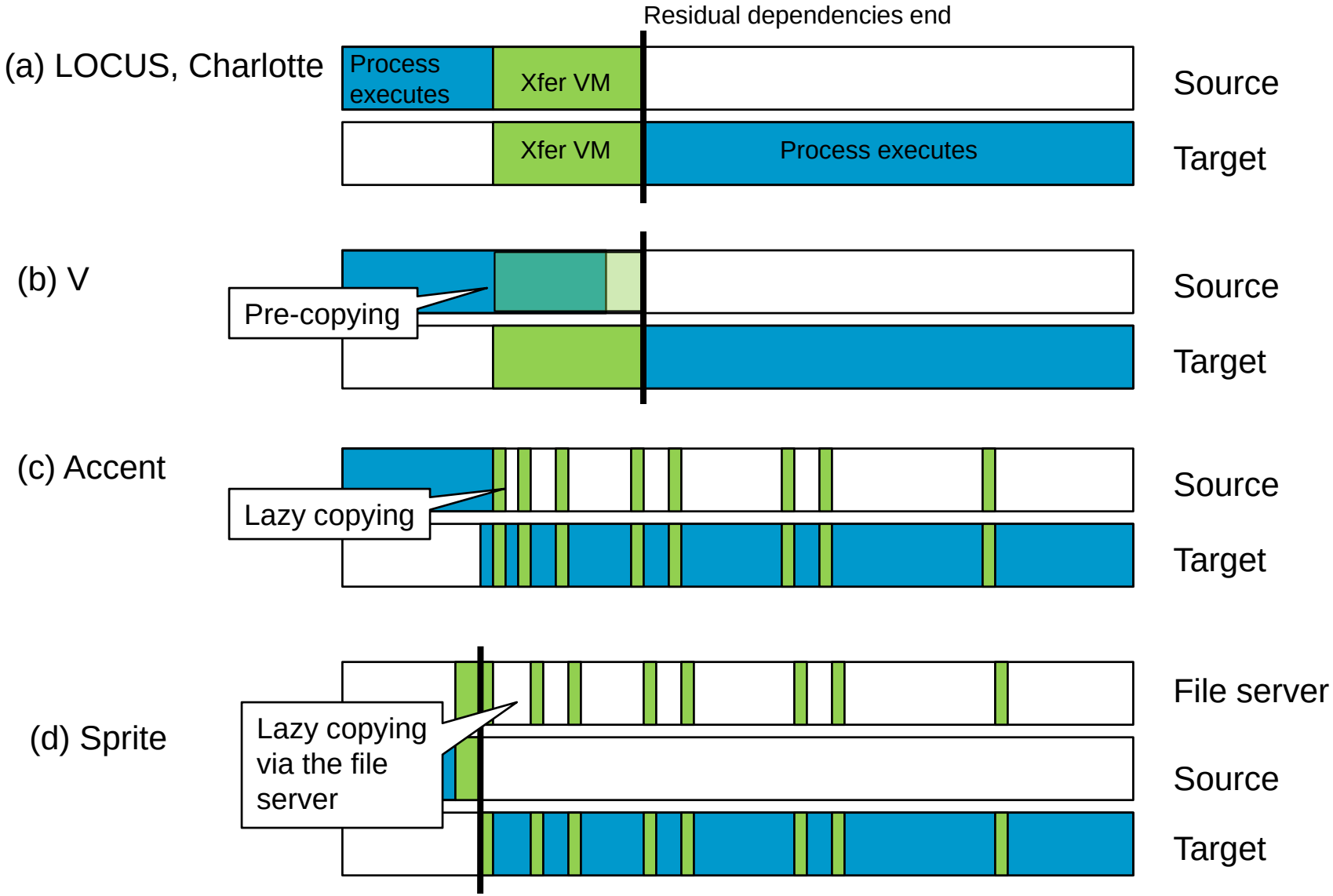
Process migration in Sprite

- Most commonly as part of the exec or when evicted
- Key goal – High transparency
  - A process behavior should not be affected by migration – A remote process has same access to VM, files, devices, and other resources as in its home machine
  - A process appearance to the rest of the world shouldn't be affected either
- Goals
  - **High transparency**
  - Avoid residual dependencies
  - **High migration and migrated process performance**
  - Low complexity, so that it is easy to maintain

The overall problem – managing state

- Migration techniques depend on associated state
  - Virtual memory – Biggest amount of state
  - Open files – In process VM and kernel, including identifier, current access point, cache file blocks
  - Message channels – In a msg-based OS, instead of open files
  - Execution state – Whatever you need for a context switch
  - Other kernel state – Process and user id, cwd, stats, …
- For each piece of state, the system could
  - Transfer the state – easy for private state
  - Arrange for forwarding – ideally done transparently
  - Ignore the state & sacrifice transparency – In Sprite, only for memory-mapped I/O devices such as frame buffers
- Transparency doesn't always make sense
  - E.g. How much mem. is there?

Mechanism for migration - VM



Mechanisms for migration – VM

- Virtual memory
  - (a – LOCUS) May be long and wasteful
  - (b – V) Precopying speeds things up, but may require copying some pages twice
  - (c – Accent) Cheap but leaves residual dependencies, potentially on every visited machine
  - (d – Sprite) A form of lazy copying
    - Source machine freezes process, flushes dirt pages to backing files, discards its address space
    - In target machine, process starts executing w/o resident pages
    - Since the server uses memory caching, disk operations may be avoided
    - Dirty pages have to be transfer twice over the network
    - Not an issue for exec transfer, but yes for later eviction
    - Things can get costly with shared writable memory, so it's disallowed

Mechanisms for migration – files and PCB

- Migrating open files – i.e. its components
  - File reference – reference is also guarantee of existence – don't close it for migration
  - Caching information – Sprite permits caching, but disables it when a process opens for writing – need modification of server code to allow open in target before closing it in origin without disabling caching (atomic operation)
  - Access position – Can also be shared among processes; similar approach to caching – disable local mgnt. and make the server responsible if it becomes shared
- PCB
  - Some state is left in the home machine as it needs to assist with some operations (e.g. `fork`)

Supporting transparency

- High transparency
  - A process behavior should not be affected by migration
  - A process appearance to the rest of the world shouldn't be affected either
- Four techniques to achieve this
  - Make kernel calls location independent – e.g. common file name space
  - Transfer state at migration time to ensure you can use normal calls
  - Forward few calls that can't be implemented transparently, e.g. `gettimeofday, getpgrp`
  - Set of ad-hoc techniques for a few kernel calls
    - E.g. `fork` – since process ids include name of home machine, the home machine allocates it

Residual dependencies

- Residual transparency – Need for a host to maintain data structures or provide functionality for a process after process migrates away from host
- Bad
  - Reliability – One host failure affects another one
  - Performance – Cost of remote operations
  - Complexity – Distributed state is harder to manage
- You can't always avoid it
  - If using a remote device, for example
  - To maintain transparency, e.g. process creation and termination
  - May even improve performance sometimes, e.g. lazy copying
- Sprite pays a bit for higher transparency

Migration policies

- Four aspects of migration policies
  - *What to migrate*
  - *When to migrate it* - Eager et al. & Harchol-Balter et al. articles on migrating active processes
  - *Where to migrate it* – This is automated, the rest is manual. A daemon process checks (lack of) activity and reports candidate host to the central migration server
  - *Who makes the above decisions* – Kernel provides no support, user level apps offer some assistance
- Using migration in Sprite
  - With `pmake` & `mig`
  - Eviction – moving foreign process back home ;)

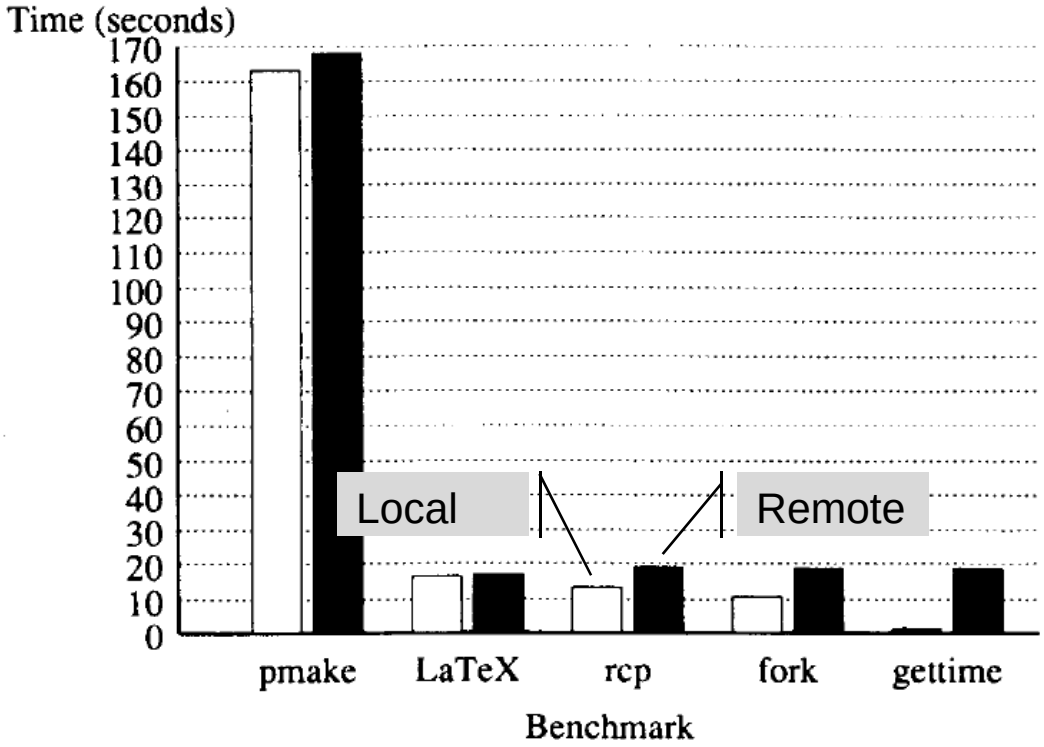
Migration overhead

Action	Time/Rate
Select and release an idle host	36ms
Migrate “null” process	76ms
Transfer info for open files	9.4ms/file
Flush modified file blocks	480KB/sec
Flush modified pages	660KB/sec
Transfer exec arguments	480KB/sec
Fork, exec null process w/ migration, wait for child and exit	81ms
Fork, exec null process locally, wait for child and exit	46ms

Cost of transferring open files is dominated by RPC latency – 3 RPC at 1ms each

Speed of transferring virtual mem. pages and file blocks is determined by RPC bandwidth – 480-660 KB/sec

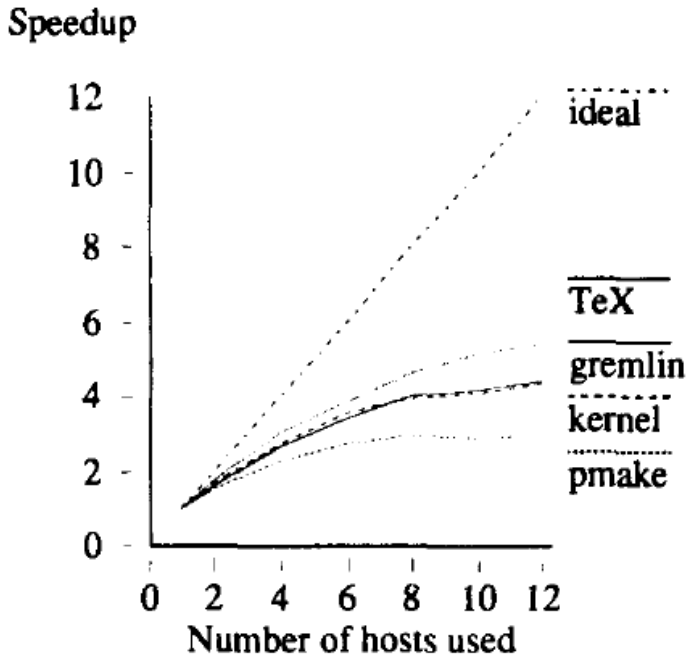
Overhead of remote execution

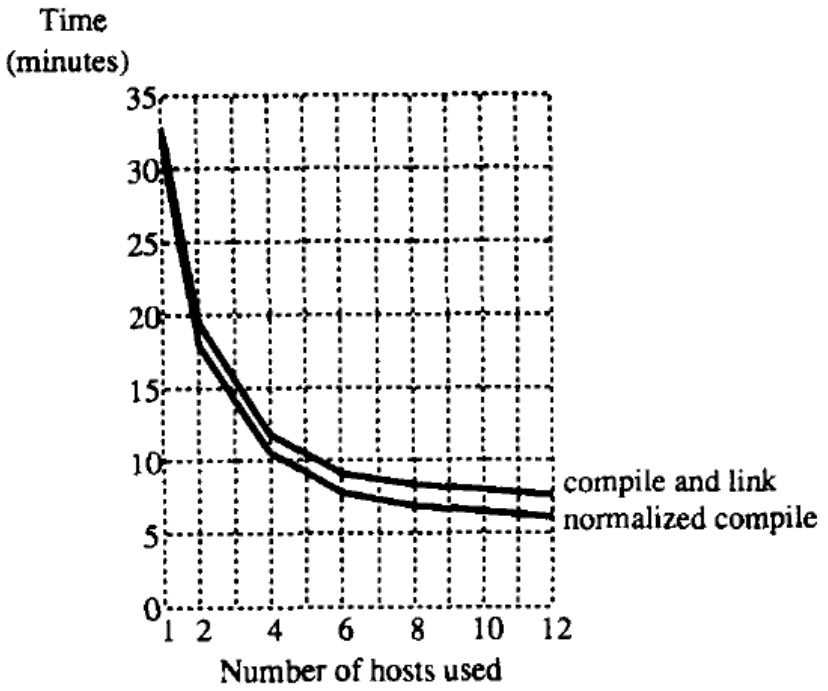


Name	Description
Pmake	Recompile pmake source sequentially using pmake
LATEX	Run Latex on a draft of this article
RCP	Copy a 1MB file to another host using TCP
fork	Fork and wait for child, 1000 times
gettime	Get the time of day 10,000 times

Application performance

Benchmark – compiling 276 Sprite kernel source files and linking the object files





More typical pmake speed-ups

Overall, the bottlenecks are the CPU in the file server and the host running pmake.

Usage patterns

- Instrumented Sprite to keep track of migration related stats – this is for a one-month period
- Remote processes – 31% (0.27-88%) of all processes
- Fraction of exec-time (full migration) 86% (14%) – so VM migration is not that important
- Host change from idle/active – 26’ in avg, 0.12 processes evicted per hour, with 25 hosts, as a result
- Hosts are 66-78% available for migration
- 86% of migration requests were fully satisfied; 2% couldn’t be cover at all

Summary

- Significant improvements to get from migration, but
- Its use may be limited by lack of (easy to use) applications
- Look at migration costs in the whole picture, not through microbenchmarks
- Migration for load balancing may not work in Sprite setting given costs of VM/file cache migration
- Make it easy to use; this facilitates adoption, forces maintenance and let's you find holes in your system