Speculative Execution

I. Background

Long tradition of research on speculation for computer architecture:

- branch prediction
- value prediction (!)

But these speculations have limited upside... what about speculation at the application level?

Insight: OS controls side effects, so OS can limit the impact of speculation!

- E.g., delay printf until the computation is really going to happen for sure
- If cancelled, we just delete it

Start with the file system: it is often synchronous for actions that work as expected such as cache hits

- we can speculate that we will hit, and then recover if we are wrong
- synchronization delays are the cost of consistency
- distributed file systems (DFS) have to choose between these delays and weaker consistency: they generally weaken consistency

NFS provides "close-to-open" consistency (weaker):

- when you open a file you will see all changes from others that have closed that file
- plus a 30-second delay!
- do not see changes until the file is closed, which reduces the communication overhead

Strict synchronization is pessimistic:

- most file operations do not conflict
- Speculator is optimistic

Basic operation:

- when the OS reaches a blocking operation
- 1) checkpoint state (to enable recovery)
- 2) speculate on the answer
- 3) continue execution immediately using this answer
- 4) when real answer comes in, either continue along, or abort and restart from the checkpoint

Speculator

II. Speculator

Key invariants:

- o Know when a process is speculative
- o Speculative processes cannot externalize output (just queue the output)
- o Track speculative operations as they affect other processes! (they become speculative too)
    - fork, exit, signals, pepes, fifos, sockets, local file, distributed files
    - other forms of IPC just block until speculation is resolved
- o No modification of applications!

Performance: 2x over local networks, 10x over wide-area networks

NFS normal behavior:

- o async writes of the data to the server
- o sync commit RPC (=> round trip plus disk write)
- o speculator guesses that all writes succeed and so does the commit
- o on open, speculator guesses that cached copy is still valid

Three characteristics:

- o ***Speculation is usually right:*** for DFS, only wrong in the case of conflicts, which are rare
- o ***checkpointing is faster than remote I/O***
    - copy-on-write fork operation is fast; most pages not copied
    - child only executes if speculation was wrong.
    - take and discard a checkpoint for a small process = 52us !
    - big process = 6ms
- o ***Spare CPU for speculation*** (low opportunity cost)

III. Implementation

Linux 2.4.21, 7500 lines of C

- o create_speculation -> spec_id
- o commit/fail calls

Creating a speculation

- o copy-on-write fork (save the child for abort)
- o save the state of open file descriptors, copy pending signals (to replay them)
- o child is not runnable

Speculator

- simply discard the child (reclaim pages) if speculation commits
- aborted processes are killed next time they try to run
- child process *assumes the identity* of the failed process
  - process id, thread group id
  - restore file desciptors and signals
  - reexcute system call that caused speculation (would it speculate again?)

Two new data structures in the kernel:
- 1) track set of kernel objects that depend on speculation (so you can reclaim them)
- 2) an undo log for each kernel object
- On a nested speculation, you can reuse the undo log (but may undo more work than necessary)
  - if more than 500ms has passed create a new undo log, so that you won't undo too much work
  - if the first operation has side effects (like mkdir), then better to create a second log. If the first operation succeeds its side effects are visible, so we don't want to undo them!

Single process correctness:
- Invariant: speculative state never visible to external device or user
  - screen output, network packets, IPC, etc.
- Invariant: a process should never see speculative state unless it is explicitly dependent on that state (or its producer).
  - If it does see the state, and that speculation fails, then this process must fail too
  - Easiest solution is to block new process until speculation resolves (but this limits parallelism)
  - Basic version: make a special syscall vector table that blocks all system calls that would externalize output
  - V2: allows syscalls that are read only
  - V3: allows syscalls that modify only private state to this process (example: dup2)
  - V4: allows syscalls to speculative file systems
  - V5: allows read/write to speculative files
  - V6: buffer write call output until speculation resolves (to screen or network)

Multi-process speculation:
- DFS: failure => invalidate cached copy
- Ramdisk: rmdir => keep around old version so you can put it back
- ext3: never write speculative data to disk (no steal!). On failure, can just delete this buffer page rather than write it out (no undo log for the disk)
- problem with no steal: some pages may never get written back (e.g. superblock)
  - solution: use redo/undo functions and keep two version (spec and nonspec). update non-spec on commits and only write it out
- reorder compound ext3 transactions so that speculative actions are postponed

Speculator

- pipes: need to track create/delete as well as read/write
- signals: can't checkpoint the receiver at an arbitrary point (why?)
  - instead: cause receiver to checkpoint soon
  - deliver a "pending" signal, but take no action
  - when process returns from the kernel, the pending signals become real and are handled
  - checkpoint is taken a this time (which is safe)
- exit: keep around pid until process exit is confirmed

IV. Using speculation

General case:

- speculate on something in cache (if not in cache, then don't speculate)
- convert sync call to verify cache entry into async call
- start speculation
- on reply to async call, decide commit/fail

Harder for mutating operations!

- side effects of speculation could become visible to others
- key solution: server need not speculate! It knows if the hypothesis is wrong and can act accordingly
- for each RPC, include list of hypotheses for server to validate

Group commit:

- speculation leads to many outstanding "synchronous" operations
- can do one write for the group, rather than one write each!
- just delay a bit and the client will send you more work, which you can amortize with one actual write
  - great for journals, does it matter for traditional file systems? (which have writes to different parts of the disk?)

V. Evaluation

Very FAST!

Rollback cost is low

Group commit helps a great deal for journaled file system (and some for NFS)