Speculations: Providing Fault-tolerance and Recoverability in Distributed Environments

Cristian Ţăpuş
California Institute of Technology
crt@cs.caltech.edu

Jason Hickey
California Institute of Technology
jyh@cs.caltech.edu

1 Introduction

Building safe and reliable programs is an important but difficult endeavor. The challenge is even greater in the context of distributed environments, which may involve complex synchronization operations in the presence of process and network failures.

Transactions are one of the earliest and simplest abstractions for reliable concurrent programming [2]. They provide fault-isolation by guaranteeing the atomicity, the consistency and the durability of the actions performed as part of the transaction. Traditional transactions also provide isolation, which prevents the independent actions inside of a transaction from being visible to the outside world until the transaction either aborts or commits.

In this paper we consider the case where multiple processes may cooperate in a transaction, using message passing for communication. We relax the transactional isolation property to permit inter-process communication while executing inside the transaction. This model can improve performance and provide fault-tolerance for distributed applications. We call these transactions with relaxed isolation *speculations*, and we introduce them as programming language primitives.

Traditional checkpointing and rollback mechanisms used to provide recoverability are also similar to our approach. However, there are a few differences, as follows. Speculations can provide programs with alternate execution paths upon rollback. Speculations are lightweight checkpoints that are stored in memory and can be coupled with real checkpointing mechanism for increased reliability. Speculations are exposed as programming language primitives that have a semantics closer to that of transactions than that of checkpoints. Our system adapts mechanisms designed for checkpointing/rollback systems [1] to ensure safe recovery lines in case of distributed speculation rollback.

While speculations are similar to the concept of lookahead-rollback introduced by the TimeWarp [3] mechanism, we extend the concept by allowing both explicit and implicit speculations through programming language extensions.

The main contributions of this paper include: (1) the introduction of a new programming model based on speculations, (2) the definition of new speculative programming language constructs for distributed applications, (3) the description of a prototype implementation of speculations in the Linux kernel where speculative operations, including distributed commit and rollback, are transparent.

2 Speculations

We define a *speculation* to be a computation that is based on an assumption whose verification may be delayed. We introduce three primitives for defining speculative execution. *Speculate* defines the entry point of a speculation. *Commit* marks the validation of the assumption on which the speculation is based on and the program continues execution as expected. *Abort* is used when the assumption on which the speculation is based is invalidated and the computation needs to be rolled back. In this case, the process is rolled back to the state it was in before entering the speculation and a different path of execution may be taken.

The three functions described above have the following types associated with them.

$$\begin{array}{lcl} \text{speculate} & : & \text{void} \rightarrow \text{int} \\ \text{abort} & : & \text{void} \rightarrow \bot \\ \text{commit} & : & \text{void} \rightarrow \text{void} \end{array}$$

The speculate function returns an integer, where the integer is zero when the speculate function is first called, and non-zero if the speculation is later rolled-back. We call the *commit branch* the execution that follows after an initial *speculate* call until either an *abort* or a *commit* call are encountered. If the speculation is aborted during the speculative execution the process is rolled back to the state it had when the speculate call was executed, and a

non-zero value is returned. In this case, the program may take an alternate execution path, as if it never executed on the *commit branch*. We call the code executed in this case the *abort branch*. The *abort* and the *commit* calls do not take any parameters, and conceptually, they don't return a value. Their main purpose is to guide the flow of the speculative program. Thus, speculations have dynamic scoping, based on when the *abort* or *commit* functions are called.

Speculative message passing enables speculative parallel computation and provides mechanisms for distributed rollback and commit. One of the key properties of distributed speculations is that the outcome of the speculation can only be decided by its owner (the process that started it). This semantics prevents unnecessary rollbacks if any of the other processes involved in the speculation fails. The semantics is sound and performs as expected in all common cases.

Since actions performed inside a speculation are not isolated, other processes can become implicitly speculative if they base their computation on a speculative message. Thus, speculations extend the traditional language-supported exception mechanisms to distributed environments by forcing implicitly speculative processes to roll back along with the initiator of the speculation. This component is one of the contributions of this model.

3 Implementation overview

We have implemented speculations as an extension to the Linux kernel (version 2.6). Our system is based on a strong formal operational semantics [4] which increases the confidence in our implementation.

Speculative primitives. The speculative primitive *speculate*, *abort*, and *commit* are implemented as three new system calls: *speculate()*, *spec_abort()*, and *spec_commit()*.

Starting a speculation. When a user process enters a new speculation, by calling the *speculate()* call, a lightweight checkpoint is created using the *fork()* system call with special parameters that create a new process with the same PID and the copy-on-write mechanism associated with fork. The new process corresponds to the abort branch of the speculation and is not executed unless the speculation is aborted.

Committing a speculation. When a speculative process that has started a speculation calls the *spec_commit()* system call it triggers two actions. The process discards the local lightweight checkpoint if it is the sole owner of the speculation. If the speculation is co-owned and the other owners have not decided the outcome of the speculation the process becomes implicitly speculative and continues execution. If the process is the last owner to decide the outcome of the speculation it broadcasts a COMMIT message through publish/subscribe channels to all processes that belong to the speculation. Upon receiving the COMMIT message lightweight checkpoints are discarded and execution continues on the same path.

Aborting a speculation. A speculation is aborted when the initiator executes the *spec_abort()* call. This triggers the broadcast of an ABORT message to the corresponding speculative group and it rolls back the process by reviving the *abort branch* created upon speculating. The recipients of the ABORT message roll back their computations as well.

Sending a speculative message. Messages sent by a speculative process are tagged with a new IP option (IPOPT_SPEC). This approach is non-invasive as far as the sending and receiving of messages are concerned.

Receiving a speculative message. Receiving and processing speculative messages required several modifications in the kernel network stack. The IPOPT_SPEC option needs to be processed and its associated information is pushed to the transport layer. This information is used to force the receiver of a speculative message to join the speculation in an implicit manner only when the data contained in the message is passed on to the user-level.

Entering, committing or aborting an implicit speculations are completely transparent to the process that has been absorbed in it. When the kernel receives notification on the outcome of a speculation it either automatically rolls back the processes that are involved in the speculation or it discards the saved checkpoints.

4 Future Work

We are currently investigating support for nested speculations inside the Linux kernel and we are developing a distributed filesystem with support for speculations, i.e. supporting rollback of files and supporting speculation propagation through files.

Full speculative support at the operating system level would allow us to develop efficient, optimistic communication protocols and new distributed software debugging tools that combine speculative execution with model checking and distributed logging mechanisms.

References

[1] DAMANI, O. P., AND GARG, V. K. How to recover efficiently and asynchronously when optimism fails. In *International Conference on Distributed Computing Systems* (1996), pp. 108–115. 1

[2] GRAY, J., AND REUTER, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1994. 1

[3] JEFFERSON, D. R. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, 3 (1985), 404–425. 1

[4] ŢĂPUŞ, C. *Distributed Speculations: Providing Fault-tolerance and Improving Performance*. PhD thesis, California Institute of Technology, Pasadena, CA, June 2006. 3