The aim of this post is to clarify some details surrounding the topic of speculative execution with the resource machine. It's not yet entirely clear which ART report these details will best fit in.

Note: the name "speculative execution" for this concept is still tentative - objections/suggestions welcome.

Introduction

In a distributed context, since resources must have a canonical controller whose ordering and authorization is required to consume them, transactions must ultimately be ordered and executed on a single controller, and they can only consume resources which that controller controls. Consequently, in order to guarantee that all resources consumed will not conflict with other transactions, the transaction author must defer resource lookup until post-ordering execution time.

However, execution (likely replicated) on a controller is more costly and less privacy-preserving than earlier execution might be, and some resource consumptions may be unlikely to conflict, either because of infrequent usage (of that resource kind), or ownership of the resource by a specific party whose authorization (e.g. a signature) is required to consume it anyways. Thus, users of the resource machine in a distributed context may often wish to perform speculative execution

: calculation of partial or full state changes prior to ordering by the controller, using the most recent resource version data known to the party performing the early execution.

Transaction functions

Intuitively, a transaction function

is a function which generates a discrete state change captured in an instance of the Transaction

data structure as defined by the Resource Machine.

Formally:

Let a speculative execution mode

be an enum of the following type:

data SpeculativeExecutionMode = None | Local | Full

Let a transaction function

be a function of the following recursive type:

type PartialEvaluation = (Tx, Maybe TxF)

type TxF = ExternalIdentity -> SpeculativeExecutionMode -> State -> PartialEvaluation

- A partial evaluation

consists of a (presumably unbalanced) transaction and a transaction function, referred to as a continuation

.

- The first parameter will be referred to as the host identity

.

For a transaction function f

to be compliant

, the following must hold:

forall i s . f i None s = (emptyTx, Just f) forall i s . f i Full s = (..., Nothing)

(emptyTx

is an empty transaction which makes no state changes)

Stated in English:

- calling the transaction function in mode "none" must be a no-op

- calling the transaction function in mode "full" must return a transaction and no continuation

The Local

mode should decide what evaluation to perform on the basis of which identity is doing the evaluation, in order to avoid conflicts later in final transaction execution. For example, resources owned by a particular identity may typically be consumed during evaluation on that identity, since only that identity can authorize their consumption.

Note: This is not a fully general structure and it will probably change. It's not yet entirely clear to me how best to generalize speculative execution, which in full form will require probability distributions over expected state conflicts.

For example:

- a user running the transaction function locally to select the right part of state would use the Local

mode

- a node simulating future execution would use the Full

mode, but discard the result after showing it to the user

- a controller would use the Full

mode, and verify/apply the resulting tx