Slotting protocol

...

January 3, 2018

1 Overview

The purpose of the protocol is to enable distributed atomic reservation of a collection of resource. For instance, two distinct agents A_0 and A_1 may request resource collections $\{r_0, r_1, r_2\}$ and $\{r_2, r_3\}$ where $r_0, r_1, r_2, r_3 \in R$. Each agent must get all or nothing - partial request satisfaction is not permitted. Every agent request should be eventually satisfied for as long as certain degenerate situations are avoided.

The principal mechanism of a solution we offer is *distributed lane.* A distributed lane is a distributed data structure made of, at first approximation, a collection of resource request pools, one per each resource,

An empty distributed lane has the following structure:

	r_0	r_1	r_2	r_3
l_0				
l_1				
l_2				
l_3				
l_4				

where columns represent resources request pools and rows represent lane. Request pools are modelled explicitly with one private pool per resource. Lane are virtual and only present at conceptual level.

Protocol

To lock a resource set an agent go through a number of steps defined by the following protocol. We put certain assumption on communication between parties: messages cannot be altered or lost. However, they can be reordered (and hence delayed).

Step 1. Request

When some agent A_i intends to lock resources $s \subseteq R$ it generates request to request pools associated with resources $\mathbf{r}$. Such requests are sent and received in no particular order and contain only agent name A_i. We define such message as $\mathsf{request}(\mathsf{A_i})$ and write $\mathsf{request}(\mathsf{A_i}) \to \mathsf{r_k}$ to state it is addressed to resource $r_k \in \mathbf{r}$.

Step 2. Reply

When a request pool r_k receives request $\mathsf{request}(\mathsf{A_i})$ it increments $\mathsf{pp}(r_k)$ and replies with message $\mathsf{reply}(\mathsf{pp}(\mathsf{r_k})) \to A_i$. Value $\mathsf{pp}(r_k)$ is the *promise pointer* of request pool r_k. The pointer is the largest *promised* index in request pool. The initial value $\mathsf{pp}(r_k)$ is unique for every request pool and is assigned statically.

Notice that addressed message $\mathsf{reply}(\mathsf{pp}(\mathsf{r_k})) \to A_i$ does not carry information about the sender.

Step 3. Confirm

After sending $\mathsf{request}(\mathsf{A_i})$, agent A_i awaits for all replies to arrive; the reply messages carry values $\mathsf{pp}(r_k)$. Once all messages are received, the agent computes $n = max_{r \in \mathbf{r}} \mathsf{pp}(r)$. This value is used to generate message $\mathsf{reserve}(\mathsf{A_i}, \mathsf{n}) \to r, r \in \mathbf{r}$, carrying maximum value n and addressed to all the pools.

Step 3-4. (Paulius)

In this version one can combine Steps 3 and 4. Since the approach of this protocol version is to iteratively try to form a lane. A basic solution could be as follows:

1. After receiving all replies one can check $\forall e \cdot e \in reply[\{A_i\}]^{-1} \Rightarrow e = n$
   a) If true proceed to Step 5a.
   b) Else prooced to Step 5b.

Remark: In order to allow $\forall e \cdot e \in reply[\{A_i\}]^{-1} \Rightarrow e \leq n$ one must make an assumption on request arriving frequency.

Step 4. Accept/Reply (Alexei)

When a request pool r_k receives message $\mathsf{reserve}(\mathsf{A_i}, \mathsf{n})$ it proceeds in one of the two ways:

1. if n is equal or greater than the current value of *promise pointer* $\mathsf{pp}(r_k)$ then the pool replies with message $\mathsf{accept}(n) \to A_i$;
2. if n is less than the current value of *promise pointer* $\mathsf{pp}(r_k)$ then the pool replies with message $\mathsf{reject}(\mathsf{pp}(\mathsf{r_k})) \to A_i$.

Step 5.a. Write (request)

Agent A_i again awaits for all replies to arrive. In a general case, these are made of $\mathsf{accept}(n)$ (all n values are same by the nature of the protocol) and reject messages. This step applies only to the situation when there are no rejections.

Since all the pools replied with the acceptance, it is now possible to write into all the pools and thus form a new lane. The agent replies with message $\mathsf{write}(A_i, n) \to r_k$ to each pool r_k.

Step 5.b. Restart

If any reply message from pools is a rejection message then the agent must re-negotiate index for a new lane. The current protocol round is abandoned and the agent starts with Step 1.

Step 6. Write

On reception of $\mathsf{write}(A_i, n)$, request pool writes A_i at index n.

Consumption

The protocol steps above describe how to form lanes from distributed pools. The pragmatics of such lanes is atomic reservation of resource sets. The lanes semantics guarantees, as we shall prove in the model, the following two properties:

1. every reserved value A_i forms a valid lane: it has same index in each request pool;
2. the smallest index of all request pools (i.e., the lane with a smallest index) define a valid set of requested resources that can be locked;
3. there are no deadlocks.