Early Scheduling in Parallel State Machine Replication

Eduardo Alchieri, Fernando Dotti, and Fernando Pedone

Universidade de Brasilia, Pontifica Universidade Católica do Rio Grande do Sul, and University of Lugano
State Machine Replication (SMR)

- Fundamental approach to fault tolerance
  - Google Spanner
  - Apache Zookeeper
  - Windows Azure Storage
  - MySQL Group Replication
  - Galera Cluster
  - Blockchain, …
SMR is intuitive and simple

same order
deterministic
execution

Clients

Servers
Key observation

- Independent requests can execute concurrently
- Conflicting requests must be serialized and executed in the same order by the replicas
- Two requests conflict if they access common state and at least one of them updates the state
Parallel State Machine Replication

- **Late scheduling**
  - Scheduling happens after requests are ordered

- **Early scheduling**
  - Scheduling decisions happen before requests are ordered
  - E.g., worker $t_x$ executes requests on X, worker $t_y$ executes requests on Y
Scheduling tradeoff

- Low Concurrency
- High Concurrency

- Low Synchronization Overhead
- High Synchronization Overhead

- Classic SMR
- Ideal
- Late Scheduling
- Early Scheduling

This paper
Our contributions

- Generalization of Early Scheduling
  - Classes of requests: expressing application concurrency
  - How to automatically map classes to worker threads
  - How the resulting technique compares to late scheduling
Classes of requests

- Readers and writers
  - Class $C_R$: read requests
  - Class $C_W$: write requests

Diagram:

- $C_R$ with external conflict
- $C_W$ with internal conflict
Mapping classes to workers

- Define workers that execute requests in the class
- Define class type
  - Sequential: one request at a time
  - Concurrent: requests executed concurrently
Early Scheduling execution model

Class C is CONCURRENT: request assigned to $t_0$ OR $t_1$

ordered requests $R_1, R_2, \ldots$ in class C

scheduler

Replica
Early Scheduling execution model

Class C is SEQUENTIAL: request assigned to $t_0$ AND $t_1$

ordered requests $R_1, R_2, \ldots$ in class C

scheduler

Replica

Class C is SEQUENTIAL: request assigned to $t_0$ AND $t_1$
Mapping classes to workers

Rule #5

If $C_1$ and $C_2$ conflict, and are sequential, then $C_1$ and $C_2$ must have one worker in common.
Mapping classes to workers

- Local reads most common requests
- Workers: $t_0, t_1, t_2, t_3$
Optimizing scheduling

- **O₁a**: Minimize workers in sequential classes
- **O₁b**: Maximize workers in concurrent classes
- **O₂**: Assign workers to concurrent classes in proportion to class weight (i.e., more work, more workers)
- **O₃**: Minimize unnecessary synchronization among classes
## Optimization model

Algorithm 3 Optimization model.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:</td>
<td>[ \forall c \in C : \forall t \in T \text{ uses}(c, t) ] // R.1</td>
</tr>
<tr>
<td>16:</td>
<td>[ \forall c \in C : [c_1, c_1] \Rightarrow \text{Seq}[c_1] ] // R.2</td>
</tr>
<tr>
<td>17:</td>
<td>[ \forall c_1, c_2 \in C : [c_1, c_2] \Rightarrow \text{Seq}[c_1] \lor \text{Seq}[c_2] ] // R.3</td>
</tr>
<tr>
<td>18:</td>
<td>[ \forall c_1, c_2 \in C, t \in T : [c_1, c_2] \land \text{Seq}[c_1] \land \text{Cnc}[c_2] \land \text{uses}[c_2, t] \Rightarrow \text{uses}[c_1, t] ] // R.4</td>
</tr>
<tr>
<td>19:</td>
<td>[ \forall c_1, c_2 \in C : [c_1, c_2] \land \text{Seq}[c_1] \land \text{Seq}[c_2] \Rightarrow \exists t \in T : \text{uses}[c_1, t] \land \text{uses}[c_2, t] ] // R.5</td>
</tr>
<tr>
<td>20:</td>
<td>[ \text{objective:} ]</td>
</tr>
<tr>
<td>21:</td>
<td>[ \text{minimize cost:} ]</td>
</tr>
<tr>
<td>22:</td>
<td>[ + \sum_{t \in T, c \in C : \text{Seq}[c]} \text{uses}[c, t] \times w[c] / w ] ] // O.1a</td>
</tr>
<tr>
<td>23:</td>
<td>[ - \sum_{t \in T, c \in C : \text{Cnc}[c]} \text{uses}[c, t] \times w[c] / w ] ] // O.1b</td>
</tr>
<tr>
<td>24:</td>
<td>[ + \sum_{c \in C : \text{Cnc}[c]} \right</td>
</tr>
<tr>
<td>25:</td>
<td>[ + \sum_{c_1, c_2 \in C : \text{Seq}[c_1] \land \text{Seq}[c_2] \land #[c_1, c_2]} \right</td>
</tr>
</tbody>
</table>

- **Described in AMPL**
- **Solved with KNitro**
Naive vs Optimized mapping

- Local reads most common requests
- Workers: $t_0, t_1, t_2, t_3$
Experimental evaluation

- Prototype in BFT-SMaRt environment
  - Early scheduling and late scheduling
  - Configured to crash failures (not BFT)
- Linked-list application
  - Single- and multi-shard deployments
  - Light, moderate, and heavy execution costs
  - Uniform and skewed workloads
Single-shard, reads, moderate
Multi-shard, mixed, moderate
http://www.inf.usi.ch/faculty/pedone/