Software Architecture for Scalable Computing Systems with Automatic Granularity Selection of Executable Code

Cloud VM

Pavlov M.A.
pavlovma007@gmail.com
NPP Satek Plus Ltd, Alexander Petrov,
Rybinsk Aviation Technical University, Russia
1. problem (2 slide)
2. main idea (1 slide)
3. VM runtime system (2 slides)
4. TVM=data system (3 slide)
5. granularity (3 slide)
6. Conclusions
World need **Scalability and Parallelism**

1) people factor

- all developers must switch to
  - lock free programming
  - maybe functional languages
  - think differently

but the **reality resists this** idiom

2) technological factor

- systems are not monolite
- we have many system building bricks
- one bricks to scale are simple
- another more difficult
- complex system scaling is a problem
Automatic parallelism for imperative code:

the full problem is:

“classic” dev =
- imperative programming
- parallelism by hand
- can’t to make software better without programmer

“FP” dev =
- not imperative programming
- possible automatic parallelism
- can configure runtime for better parallelism

real word problem
easy map to OOP
world resist this way
capacity limit is reached

“cloud vm / paas” dev =
- imperative programming
- possible automatic parallelism
- automatically and manual selection the better execution method

this vm not exists *
grid/MPI/etc - is not VM
Main Idea - more work with algorithm

2 equal point of view to our solution:

main purpose:

1) of course to make a work of VM
2) **in same time to make algorithm better**

collect runtime information and optimize execution:

- make more parallel,
- select better granularity
- extract all correct dataflow, find competent execution method

S - structure of solution - “what we want to compute” (dataflow)

M - methods of solution - “how we can compute better”
VM runtime system

Classic VM

- instruction flow
- Runtime processor
- data system

Cloud parallel VM

- scalable solution knowledge (CodeBase + PMS)
- need correct coordination and control flow
- scalable data system (TVM * based, see next slide)
VM subtransactions, about coordination

run pieces of code in **subtransactions**

---

the fragments are run:

- in separate OS thread
  - “A” - this process
  - “B” - this box
  - “C” - this rack
  - “D” - this room
  - “E” - this DC
  - ...etc

- in parallel, each with other

- sequentially, in one thread

---

the granularity question:

- to extract the “code block” in other, parallel fragment (async) or to run it in current thread ?
- how big the “code block” ?

---

we need statistics before answer
# TVM = Data system

TVM - the building brick of scalable data system:

<table>
<thead>
<tr>
<th>Normal access</th>
<th>Conflict access</th>
<th>Conflict detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1....R2</td>
<td>R2....W1</td>
<td></td>
</tr>
<tr>
<td>W1....W4</td>
<td>W4....W1</td>
<td></td>
</tr>
</tbody>
</table>

TVM (transaction value manager) - operation time

- storage, mutable, **CAP**
- normal access
- conflict access
- conflict detection

**store:**
- scalars
- tables, like lua

**operations:**
- set scalar, set table entry
- get scalar, get table entry
- rollback subtr=(tr,dam)
- 2 phase paxos commit

**main in params:**
- valueid
- tr=transaction
- dam=data access marker
- optional: key, value,
TVM Node implementation C++

The performance tests

- in memory
  
  write speed: \textbf{1171429} ops/sec  
  read speed: \textbf{458935} ops/sec  

- persistent (backends: leveldb, Hyper Leveldb, rocksdb)
  
  write speed: \textbf{9762} ops/sec  
  read speed: \textbf{24251} ops/sec  

TVM scaling ways

- client side “DHT” (< 500 nodes)
- cloud DT based service layer (>500-s nodes)

see next slide about scale
Distributed version of TVM

simple way (< 500 machines):

hash_func(value_id, tr) -> node_id

cloud way (>500 machines):

hash_func(value_id, tr) -> nodeid

Cloud DHT layer (k->v|doc)

scalable concurrent Tree structure service layer (in develop)

scalable TVM service layer (in develop)

Workers with fragments of code
The problem:
There are several children unit of code in every level of code context. One need to find all combination of the child unit and a processor for every parent code unit on upper level.

The ways to solve NP problems:

<table>
<thead>
<tr>
<th>the ways:</th>
<th>our steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximation</td>
<td>in stat we can aggregate values (avg)</td>
</tr>
<tr>
<td>Randomization</td>
<td>PMS makes an optimization in random place of target set</td>
</tr>
<tr>
<td>Restriction</td>
<td>PMS analyze stat. and params of fragment too</td>
</tr>
<tr>
<td>Parameterization</td>
<td>PMS make an attempt to probe several strategies for run</td>
</tr>
<tr>
<td>Heuristic</td>
<td>PMS implements different heuristic</td>
</tr>
</tbody>
</table>

knowledge, obtained by execution, help to solve the problem
granularity, the optimisation

- Performance
- Optimum
- Max parallel
  - High overhead
- Granularity
  - Increase

Find parallel traces

- Merged code
- Splitted code
  - "Anti" parallel = find sequential traces

1 processor

- N
- N
- N

3 processors

- w1
- w2
- w3
- N - Control flow

The same time

- 2 processors
  - w1
  - w2
  - N
  - Control flow

- 2 processors
  - w1
  - w2
  - N
  - Control flow
granularity, TDO, executor classes

Time
TDO = -------------- (Div)  
Overhead

overhead  execution time  
full time

Fragment’s TDO versus Executor class

TDO small  TDO big

A                B                C                D                E
only in  to run  to run  any machine  on any machine
this worker  on any  on any  in this room  in other
and  core of  machine in  of DC  DC
current  current  this rack
core  machine
Conclusions

1. Work is proceeding according to plan, ok.

2. TVM performance are good, and we know how to scale TVM

3. We have doubts: TVM are mutable storage, but immutable storage could be scale better.

4. Our VM are transactional and consistent. But in system with rollbacks we have unusual I/O system (cache reads, execute all write only in last commit phase, if rollback - no writes)