Current state of (distributed) TLC

Markus A. Kuppe

M.Sc. Student
Department of Computer Science
University of Hamburg

International Workshop on the TLA+ Method and Tools, 2012

Outline

Outline

A distributed model checker algorithm (master)

Data: SQ, FPS, $TRACE$, ϕ, $WORKER$, n

begin
    $SQ \longleftarrow initStates()$; `// Generate init states once`
    foreach $w \in WORKER$ **do** concurrently
        while $SQ \neq \emptyset$ **do**
            $S \longleftarrow subset(SQ, n)$; `// Worklist size n`
            $Succ \longleftarrow successors(w, S)$; `// remotely`
            $SQ \longleftarrow SQ \setminus S$; `// Mark states` S `done`
            if *isViolation(Succ)* **then**
                $SQ \longleftarrow \emptyset$; `// End`
                return *path(s', TRACE)*, ϕ ; `// Path to` s'
            end
            $SQ \longleftarrow SQ \cup Succ$; `// Add new succ. to` SQ
            $append(TRACE, Succ)$; `// Maintain` $TRACE$
            $H \longleftarrow hashes(Succ)$; `// Prev. calculated`
            $addToSegment(FPS, H)$; `// Into corresp.` FPS
        end
    end
end

A distributed model checker algorithm (worker)

```
Data: φ, FPS
begin
    Succ ⟵ ∅
    foreach s ∈ States do
        s′ ⟵ genSucc(s) ;                  // Generate succ. states
        Succ ⟵ Succ ∪ {hash(s'),s',s} ;    // Calculate hashes
    end
    foreach h ∈ segment(FPS, h) do concurrently check known
        Succ ⟵ Succ \ {h,s',s} ;           // Remove known states
    end
    foreach s′ ∈ Succ do check safety props
        if violates(s, s', φ) then
            signalViolation(s′, φ) ;                          // End
        end
    end
    return Succ
end
```

Outline

Basic topology

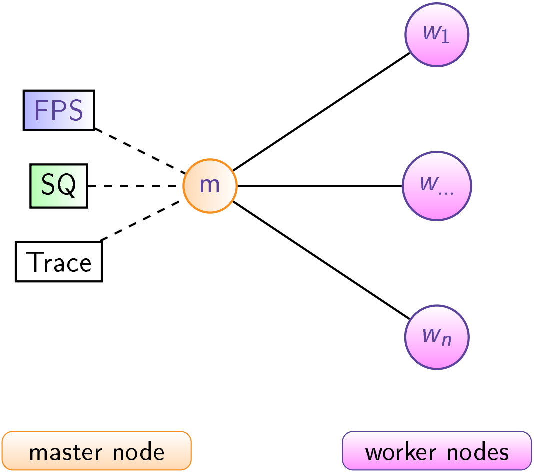

Figure: basic setup

Advanced topology

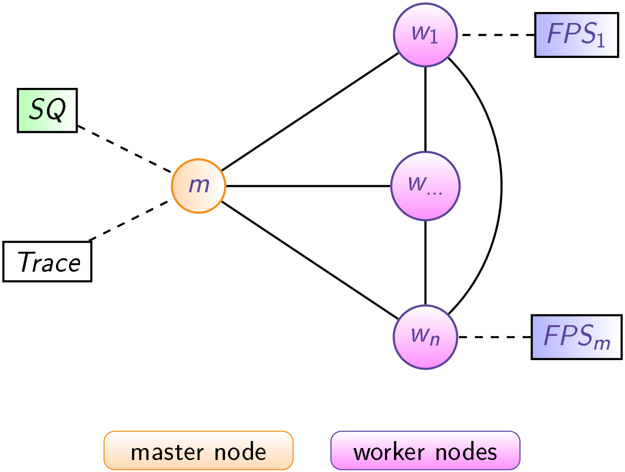

Figure: advanced setup

Outline

Fault tolerance

- 1...n workers (w)
- 1...m fingerprint sets (FPS)
  - Lost fingerprint set means corresponding states will be re-explored
  - FP collision probability will be off
- Can neither compensate loosing SQ nor $Trace$ (yet)
  - Chkpt only provides fault tolerance against program errors
  - Workaround: Keep remote backups of .chkpt files

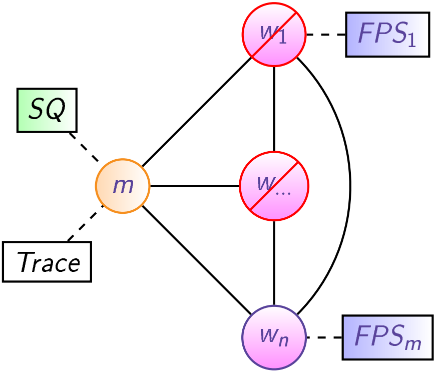

Figure: broken setup

Outline

Outline

Does (distributed) TLC perform?

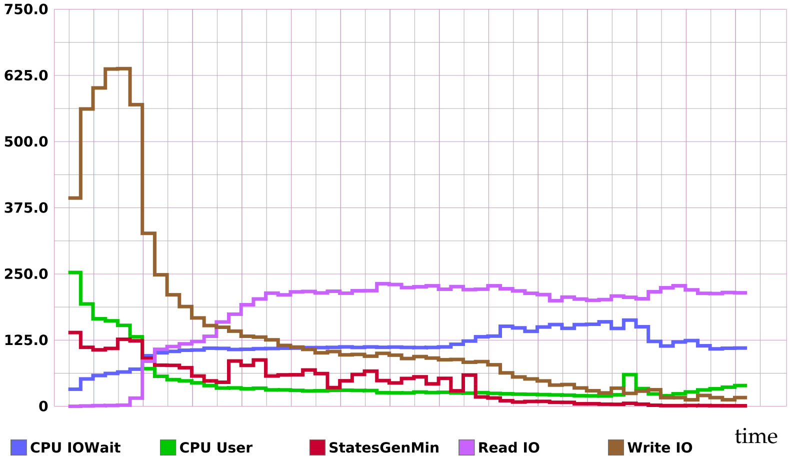

Figure:
EC2-69-xmxms60G-w32-nodes08-cDef-l1_n32_k2-RHead-fpmem08

Does (distributed) TLC perform?

- Problems
  - Performance degrades as soon as TLC goes to disk (expected)
    - I/O bound
    - (Solid state) disks order of magnitude slower compared to RAM
    - vs. much greater storage size
  - FPS memory utilization is suboptimal

Outline

Big Memory

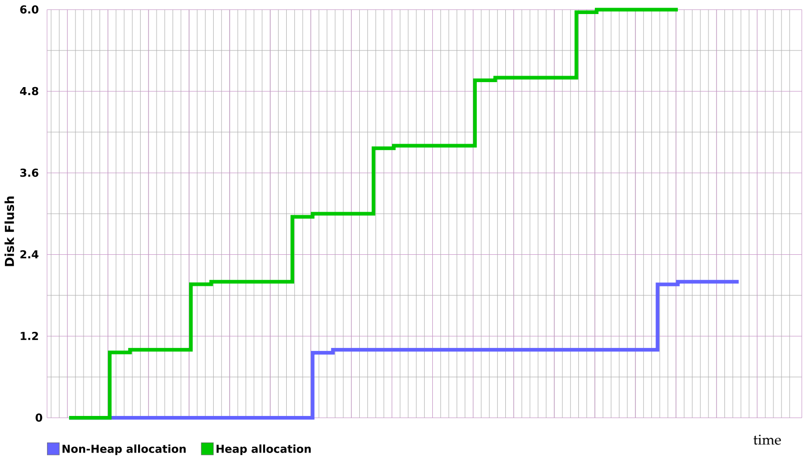

Figure: Heap and non-heap

Big Memory contd.

Allocates on heap & LSB

- long[][] (multidimensional)
  - 22% initial overhead
  - Length, class schema, pointers
- long[tblCnt] as temporary sort array during disk flush
  - 50% storage overhead
  - Sorting overhead
  - int addressing hard limit for (a single) DiskFPSet
- Sums up to net efficiency approx. 40%
- Exposed to GC
  - Fingerprints cannot gc'ed

Allocates on non-heap & MSB

- Replaces multidim. array with static continuous memory
  - No overhead
  - Initial bootstrap cost to statically allocate
- Half memory consumption by removing long[tblCnt] array completely
  - Presort in-memory FP based on most significant bits (MSB)
  - Requires on-heap 2nd level collision bucket (fix by e.g. re-probing)
- Removes GC cost completely

FPSet concurrency

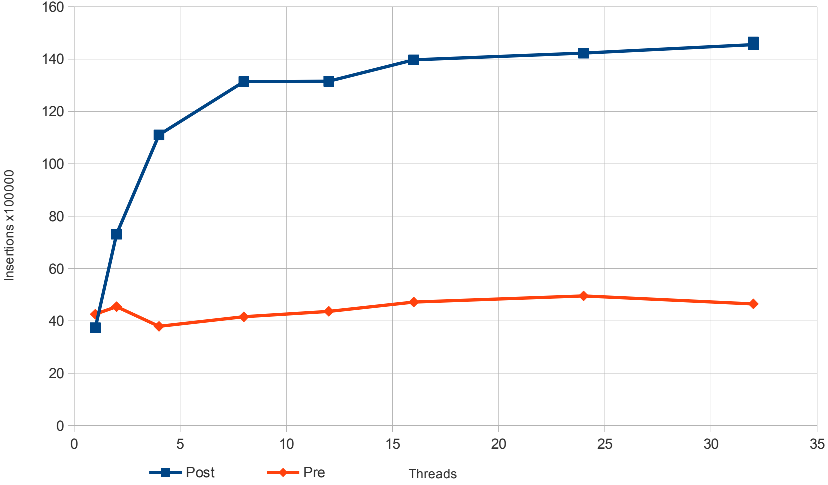

Figure: FPS memory ops concurrency pre and post e. (16 virtual cores)

FPSet concurrency contd.

- Concurrent memory read access (exclusive writes)
- Striped locks to increase concurrency/fine grained locking
  - Only lock corresponding part of hash table during memory writes
- Disk locking remains untouched (I/O is dominant cost)

Outline

Scale horizontally - Distributed TLC

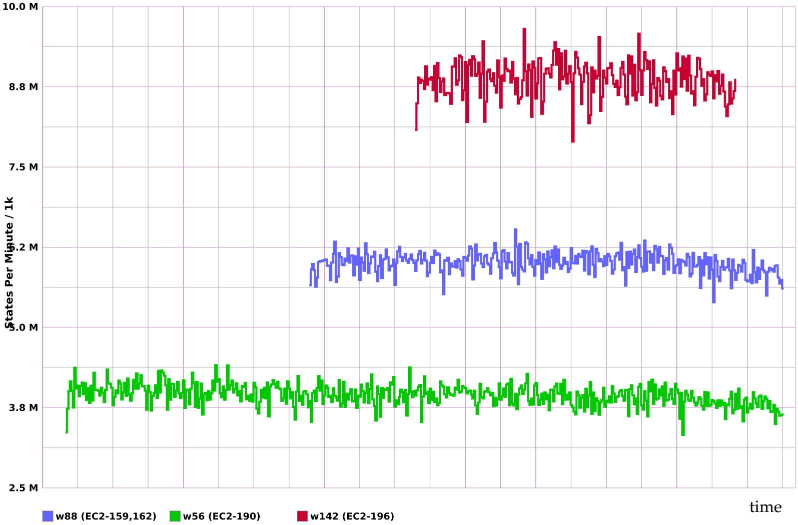

Figure: 56/8, 88/12 and 142/19 workers/nodes (l12_n10_k8)

Scale horizontally - Enhancements

- Distinguished FPS!!!
  - Remote memory still faster compared to local (solid state) disk
- Distributed FPS put and contains ops concurrently
- “BlockSelector” based on network stats to assign big chunks of work at once
  - Degraded breadth-first search
- Calculate fingerprint collision probability concurrently during end-game phase
  - Full pass over all fingerprint sets
- Node-local worker cache (1MiB) keeps 5 to 10% lookups from fingerprint sets
- (Ordered put and contains to reduce page seeks)
  - Sort executed on worker

Outline

Performance comparison

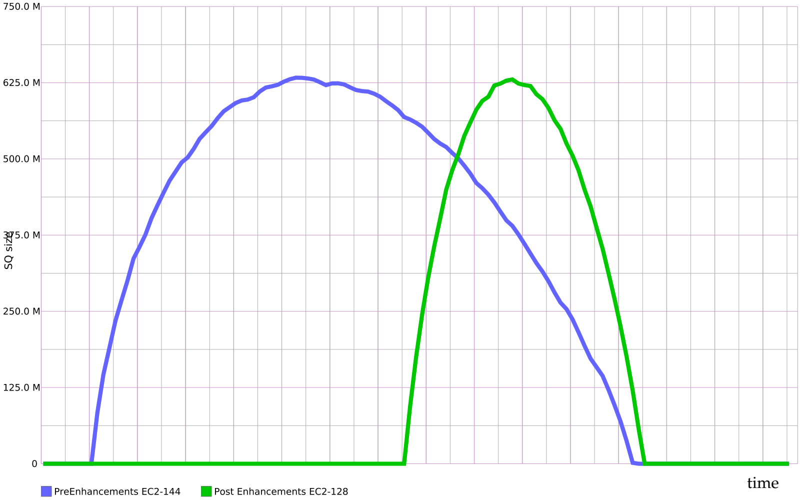

Figure: Pre and post e. *SQ* size over time

Summary

- Increased TLC performance
- Made distributed TLC scale to many machines (primarily due to distributed FPS)
  - Preliminary results show approx. 0.7 scalability factor
  - Tests need to be extended to higher node counts
- Toolbox based distributed deployment

Outline

Outlook

- Dynamic distributed FPS and bug free recovery
- StateQueue & Trace scaling and fault tolerance
- "AutoScaling" based on actual machine load
- Increased locality by partitioning/segmenting on state properties different from fingerprints
- Better support for large scale deployments (based on Jenkins scheduler?)
- Mathematical performance model

Acknowledgment

- Microsoft Research & MSR Inria Joint Lab
- Amazon AWS
- Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies - https://www.grid5000.fr
- Dr. Cliff Click "Scaling Up a Real Application on Azul" 2006 (JavaOne) - http://www.stanford.edu/class/ee380/Abstracts/070221_J1_ScalingUp.pdf

Q&A

Thank you for your attention

References

- eMail: `mailto:tla-workshop-2012@lemmster.de`
- Slides: `https://www.lemmster.de/uploads/CurrentStateDistributedTLC_MarkusAKuppe.pdf`
- tlc-perf repository: `https://github.com/lemmy/tlc-perf`
- memopt builds: `http://tla.msr-inria.inria.fr/kuppe/memopt/`

Additional material

Test model parameters

l	n	k	distinct states (k^n)	size (MiB)	cost/state (2^l)
10	6	8	2^{18} (262.144)	2	1024
10	8	8	2^{24} (16.777.216)	128	4096
10	10	8	2^{30} (1.073.741.824)	8.192	16.384
12	6	8	2^{18}	2	1024
12	8	8	2^{24}	128	4096
12	10	8	2^{30}	8.192	16.384
14	6	8	2^{18}	2	1024
14	8	8	2^{24}	128	4096
14	10	8	2^{30}	8.192	16.384
1	32	2	2^{32} (4.294.967.296)	32.768	2
1	33	2	2^{33} (8.589.934.592)	65.536	2

SQ buffer size

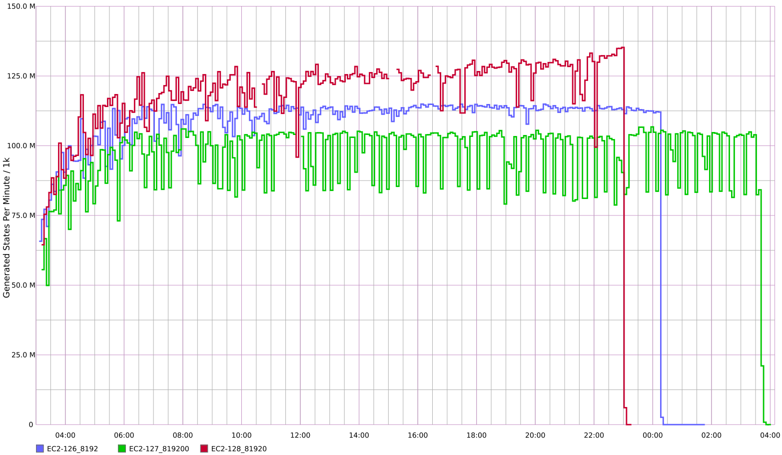

Figure: SQBufferSize

Ordered fingerprint operations

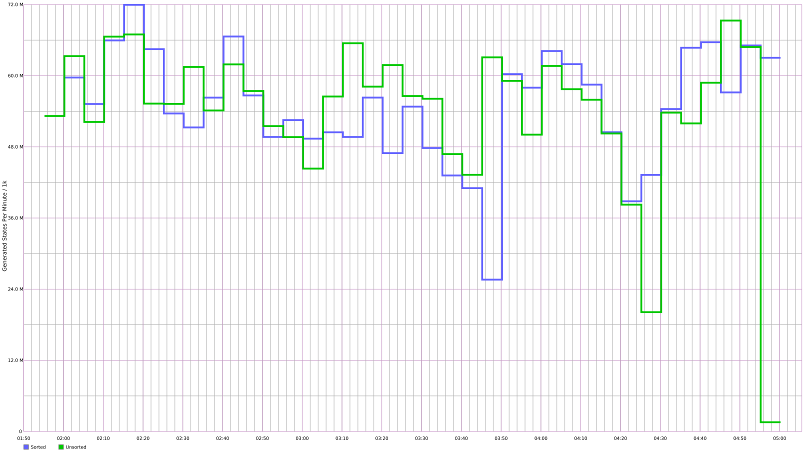

Figure: Ordered FP ops