AN ELASTIC MULTI-CORE ALLOCATION MECHANISM FOR DATABASE SYSTEMS

SIMONE DOMINICO 1, JORGE A. MEIRA 2, MARCO A. Z. ALVES 1, EDUARDO C. DE ALMEIDA 1

FEDERAL UNIVERSITY OF PARANÁ, BRAZIL 1, UNIVERSITY OF LUXEMBOURG 2
OUTLINE

- Introduction
- NUMA effect in OLAP
- PetriNet Multi-Core Allocation Mechanism
- Evaluation
- Conclusion
INTRODUCTION
NON-UNIFORM MEMORY ACCESS (NUMA)

- Avoid memory contention in multi-core machines
- Node: Portion of memory and associated processor
NON-UNIFORM MEMORY ACCESS (NUMA)

- Avoid memory contention in multi-core machines
- Node: Portion of memory and associated processor
- Fast \textit{local} RAM and slow \textit{remote} RAM
NON-UNIFORM MEMORY ACCESS (NUMA)

- Threads may run in any core over time
NON-UNIFORM MEMORY ACCESS (NUMA)

- Threads may run in any core over time
- Data is mapped at thread startup
NON-UNIFORM MEMORY ACCESS (NUMA)

- Threads may run in any core over time
- Data is mapped at thread startup
- Remote access through the interconnection
NON-UNIFORM MEMORY ACCESS (NUMA)

- Threads may run in any core over time
- Data is mapped at thread startup
- Remote access through the interconnection
- Migration: load balance, reduce intercon. usage
NUMA EFFECT IN OLAP
PARALLEL QUERY PROCESSING IN NUMA

```
SELECT *  
FROM A, B  
WHERE A.ID = B.ID  
AND B.DATE >= '1994-01-01'
```
PARALLEL QUERY PROCESSING IN NUMA

```
SELECT * FROM A, B WHERE A.ID = B.ID AND B.DATE >= '1994-01-01'
```

- Parallelism is baked in the query plan at planning time
- Batch work is assigned to threads statically (tuples or columns)
- Ex. VOLCANO and MONETDB

NUMA obliviousness through memory mapping
M. Gawadi, M. Kersten
Damon@Sigmod, 2015
NOT NUMA-FRIENDLY PARALLELISM

- TPC-H Query 6 on 1GB database
- 4-node Quad-Core AMD Opteron 8000 Series (64Gb RAM/Node)
- MonetDB (v11.25.5)
**NOT NUMA-FRIENDLY PARALLELISM**

- TPC-H Query 6 on 1GB database
- 4-node Quad-Core AMD Opteron 8000 Series (64Gb RAM/Node)
- MonetDB (v11.25.5)
IMPACT OF REMOTE ACCESS

- TPC-H Q6: SQL vs. C language (raw performance)
IMPACT OF REMOTE ACCESS

- TPC-H Q6: SQL vs. C language (raw performance)

- The OS isn’t, either
  - Not NUMA-friendly processing

Interconnect Traffic MB/s ($10^3$)

Users

- Dense/C
- Sparse/C
- OS/C
- MonetDB
IMPACT OF REMOTE ACCESS

- TPC-H Q6: SQL vs. C language (raw performance)

- Specific cores are allocated

- Threads pinned to cores
IMPACT OF REMOTE ACCESS

GOAL:
MITIGATE THE DATA MOVEMENT HANDING OUT TO THE OS THE LOCAL OPTIMUM NUMBER OF CORES IN SPECIFIC NODES
IMPACT OF REMOTE ACCESS

- SQL query vs. C language (raw performance)

CHALLENGES:

- LOCAL OPTIMUM NUMBER OF CORES
- CPU-CORE ALLOCATION

Users
PETRINET MULTI-CORE ALLOCATION MECHANISM
OVERVIEW OF OUR MULTI-CORE ALLOCATION MECHANISM

1 Monitoring threads

2 Monitoring usage
OVERVIEW OF OUR MULTI-CORE ALLOCATION MECHANISM

1. Monitoring threads
2. Monitoring usage
3. Priority allocation
OVERVIEW OF OUR MULTI-CORE ALLOCATION MECHANISM
LOCAL OPTIMUM NUMBER OF CORES

\[ \forall w \in n_{alloc} | (th_{min} < u < th_{max}) \land p(n_{alloc}) \geq p(n_{total}) \]
LOCAL OPTIMUM NUMBER OF CORES

∀ w ∈ n_{alloc} | (\text{th}_{min} < u < \text{th}_{max}) \land p(n_{alloc}) \geq p(n_{total})
LOCAL OPTIMUM NUMBER OF CORES

∀ \ w \ \in \ n_{alloc}\ |\ (th_{min} < u < th_{max}) \land \ p(n_{alloc}) \geq p(n_{total})
LOCAL OPTIMUM NUMBER OF CORES

∀ w ∈ n_{alloc} | (th_{min} < u < th_{max}) ∧ p(n_{alloc}) ≥ p(n_{total})
PETRINET MULTI-CORE ALLOCATION MECHANISM

(DEFINITIONS)

- Places={Stable, Idle, Overload, Provision, Checks}
- Transitions={t0, ..., t7}, (i.e., conditions)
- Arcs=<pi, tj > or <tj , pi>, (i.e., I/O functions)
OVERLOADED STATE

Monitoring

State Transition

CPU (%)

Check

Overload

Provision

n_alloc < 16

n_alloc == 16

n_alloc

n_alloc

n_alloc

99

3

0

25

50

75

100

0

2

4

6

8

# of Cores
OVERLOADED STATE

Overloaded state

State Transition
OVERLOADED STATE

- Action: allocate cores
ALLOCATION MODES

- **Sparse**
  - Node 0
    - Time
      - 0
  - Node 1
    - Time
      - 4
  - Node 2
    - Time
      - 8
  - Node 3
    - Time
      - 12

- **Dense**
  - Node 0
    - Time
      - 0
  - Node 1
    - Time
      - 4
  - Node 2
    - Time
      - 8
  - Node 3
    - Time
      - 12

- □ allocated
- ○ next to allocate
ALLOCATION MODES

Sparse

Dense
ADAPTIVE MODE

Priority queue

- Release

Allocate

# of DB Thread Pages

/proc/[pid]/numa_maps

allocated

next

Node 0
0 1
2 3
8 9
10 11

Node 1
4 5
6 7
1 13
14 15

Node 2
Node 3
ADAPTIVE MODE

Priority queue

Allocate

Release

allocated

next

# of DB Thread Pages

/proc/[pid]/numa_maps

ADAPTIVE MODE

Allocate

Release

allocated

next

# of DB Thread Pages

/proc/[pid]/numa_maps
ADAPTIVE MODE

Priority queue

Allocate

Release

# of DB Thread Pages

/proc/[pid]/numa_maps

allocated

next

Node 0

Node 1

Node 2

Node 3

Allocate

next
EXPERIMENTS
SETUP

- TPC-H 1GB and 100GB database up to 256 users
- 4-node Quad-Core AMD Opteron 8000 Series (64GB RAM/Node), Agr. Bandwidth 41.6 Gb/s, L3 (6MB)
- Debian Linux 8 “Jessie”
- Prototype in C language
- Database systems:
  - MonetDB (v11.25.5)
  - NUMA-Aware Microsoft SQL Server (v2017 Dev. RC2)
IMpact of the MIGRation of Threads

(TPC-H Query 6, 1 User, 1 GB Database, MonetDB)
CONCURRENT EXECUTION OF TPC-H

(256 USERS, 1 GB DATABASE, MONETDB/SQLSERVER)
CONCURRENT EXECUTION OF TPC-H

(256 USERS, 1 GB DATABASE, MONETDB/SQLSERVER)
NUMA-FRIENDLINESS: INTERCONNECTION RATIO

(256 USERS, 1 GB DATABASE, MONETDB, TPC-H)
NUMA-FRIENDLINESS: INTERCONNECTION RATIO

(256 USERS, 1 GB DATABASE, SQL SERVER, TPC-H)
AN ELASTIC MULTI-CORE ALLOCATION MECHANISM FOR DATABASE SYSTEMS

CONCLUSIONS
CONCLUSIONS

- Abstract model for the allocation of cores
- Thread scheduling works better with less cores (local optimum number of cores)
- Less thread migrations with less data movement: up to 3.87x of reduction on traffic ratio
- Up to 26% on energy savings
THANKS!

EDUARDO@INF.UFPR.BR

ACKNOWLEDGMENTS: FUNDAÇÃO ARAUCARIA, CAPES, CNPQ AND FUNDO NACIONAL DE DESENVOLVIMENTO DA EDUCAÇÃO (FNDE).
NUMA-FRIENDLINESS: INTERCONNECTION RATIO

(MONETDB, 256 USERS, 1 GB DATABASE, TPC-H)
NUMA-FRIENDLINESS: INTERCONNECTION RATIO

(SQL SERVER, 256 USERS, 1 GB DATABASE, TPC-H)

Execution Speedup (Adaptive Mode)
ENERGY SAVINGS
(MONETDB, 256 USERS, 1 GB DATABASE, TPC-H)

26%

Energy (J) 10^3

OS scheduler/MonetDB

Adaptive

CPU

HT

(a)

(b)
ENERGY SAVINGS

(SQL SERVER, 256 USERS, 1 GB DATABASE, TPC-H)

Energy (J) \times 10^3

OS scheduler/SQL Server

Adaptive

CPU

HT

11%
PETRINET MULTI-CORE ALLOCATION MECHANISM

(DEFINITION: LOCAL OPTIMUM NUMBER OF CORES)

∀ w ∈ n_{alloc} | (th_{min} < u < th_{max}) ∧ p(n_{alloc}) ≥ p(n_{total})

- w is the current workload of the database threads
- n_{alloc} is the number of allocated CPU cores (n_{alloc} ≤ n_{total})
PETRINET MULTI-CORE ALLOCATION MECHANISM
(DEFINITION: LOCAL OPTIMUM NUMBER OF CORES)

\[ \forall w \exists n_{alloc} \ | \ (th_{min} < u < th_{max}) \land p(n_{alloc}) \geq p(n_{total}) \]

- \( u \) is the average resource usage of database threads between minimum and maximum thresholds.
- \( n_{total} \) is the number of available CPU cores.
- \( p(x) \) is the performance function, where \( x \) assumes the number of CPU cores \( n_{alloc} \) or \( n_{total} \).