Data-Parallel Computing Meets STRIPS

Erez Karpas, Tomer Sagi, Carmel Domshlak, Avigdor Gal, Avi Mendelson, and Moshe Tennenholtz
Technion-Microsoft Electronic Commerce Research Center

Motivation

- Declarative query processing and user defined functions do not play well together
  - User must specify some base execution plan
  - Which optimizations are safe?

DPPS

- Framework is based on tracking *data chunks*
- Each data chunk d is associated with the amount σ_d of memory it requires

A DPPS Task is described by:

- D — possible data chunks, with sizes σ_d
- N — computing units, with memory κ_n
- A — computation primitives, each described by:
  - $\bar{I} \subseteq D$ — required input
  - $\bar{O} \subseteq D$ — produced output
  - $C : N \to \mathbb{R}^{0+}$ — computation cost
- $T : N \times D \times N \to \mathbb{R}^{0+}$ — data transmission cost
- s_0 — the initial state of the computation
- G — the goal of the computation

- A DPPS state specifies which processor holds which data chunks
- A solution is a sequence of compute / transmit / delete data actions which achieves the goal from the initial state

- The possible data chunks D and computations A may be given explicitly or described implicitly
- If they are described implicitly the sets could be infinite

Theoretical Properties

Expressivity:
☺ DPPS is at least as expressive as relational algebra with aggregation

Complexity:
☹ Optimal data-parallel program synthesis is NP-hard, even under severe restrictions
☹ Satisficing data-parallel program synthesis is NP-hard

☺ Satisficing data-parallel program synthesis with no memory constraints can be solved in polynomial time, when the possible data chunks are given explicitly.

Example: Multi Histogram

- Suppose we have a users table `T` with 10^9 users
- We want two histograms of `T`: by age and by relationship status

In SQL or similar:

```
SELECT COUNT(T.age) FROM T;
SELECT COUNT(T.rls) FROM T;
```

Query Execution Plan:

```
Agg(age, scan(T))
Agg(rls, scan(T))
```

- Suppose we have a user-defined function, `DAgg`, which aggregates by two fields simultaneously

Query Execution Plan using DAgg:

```
DAgg(age, rls, scan(T))
```

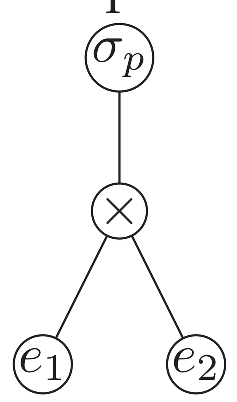
- How do we come up with this execution plan?

Compilation to STRIPS

- When computations/data chunks explicit:
  - Predicate `holds(?node, ?data)`
  - Actions
    - `compute(?node, ?computation)`
    - `transmit(?node, ?data, ?node2)`
    - `del(?node, ?data)`
  - Capacity constraints = numerical fluents

- When the computations/data chunks implicit, compilation is still possible sometimes
- When data chunks have a structure (e.g., expression trees), it is possible to represent such trees using predicates

Expression Tree   Encoding



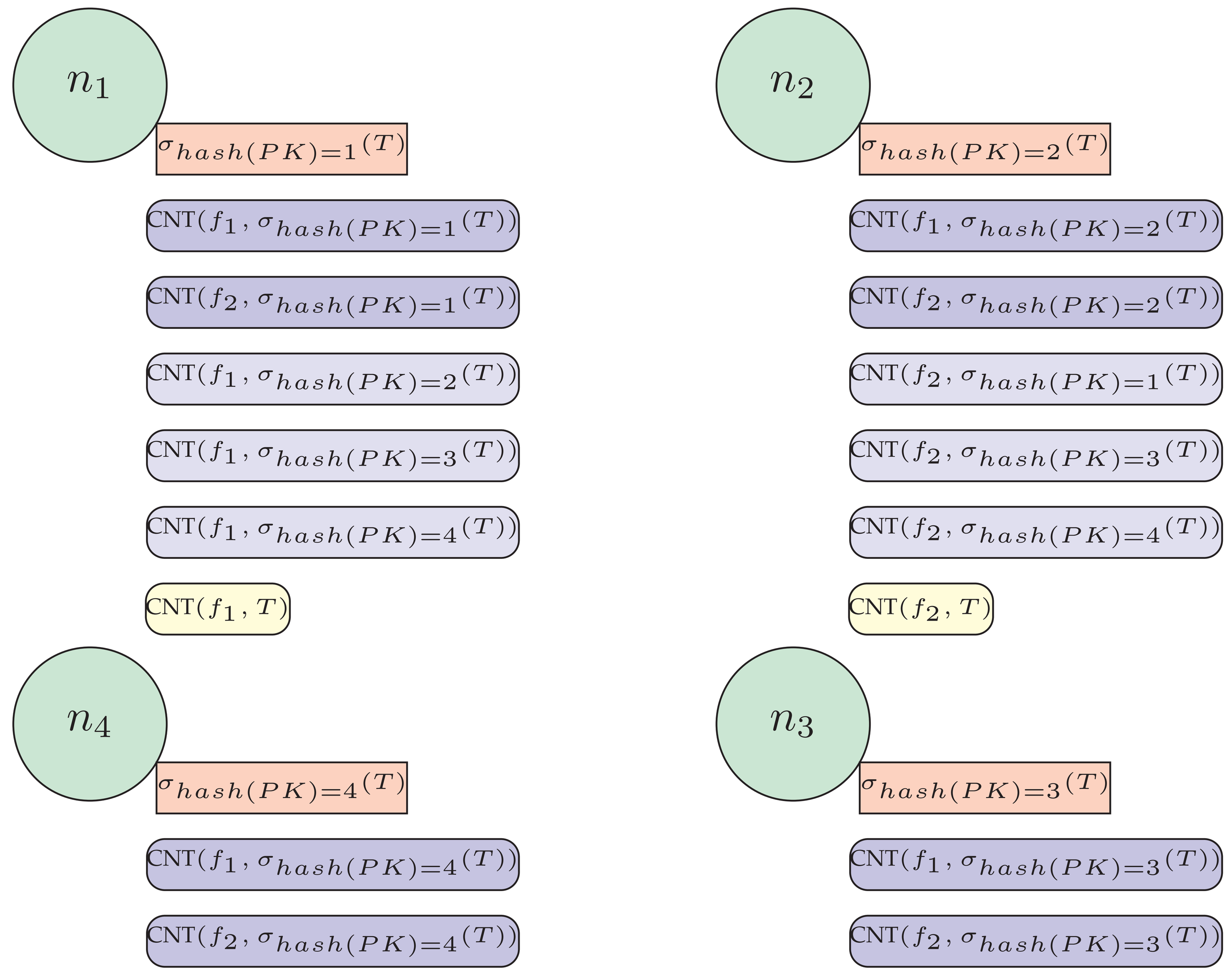
select(n_1, p, n_2)
join(n_2, e_1, e_2)

- Equivalence rules typically have limited depth, and can be encoded as operators
- More details in the paper

Empirical Proof of Concept

- Histogram of F fields of a table divided across N processors

Example for $N = 4$, $F = 2$



Query Execution Plan without `Dagg`

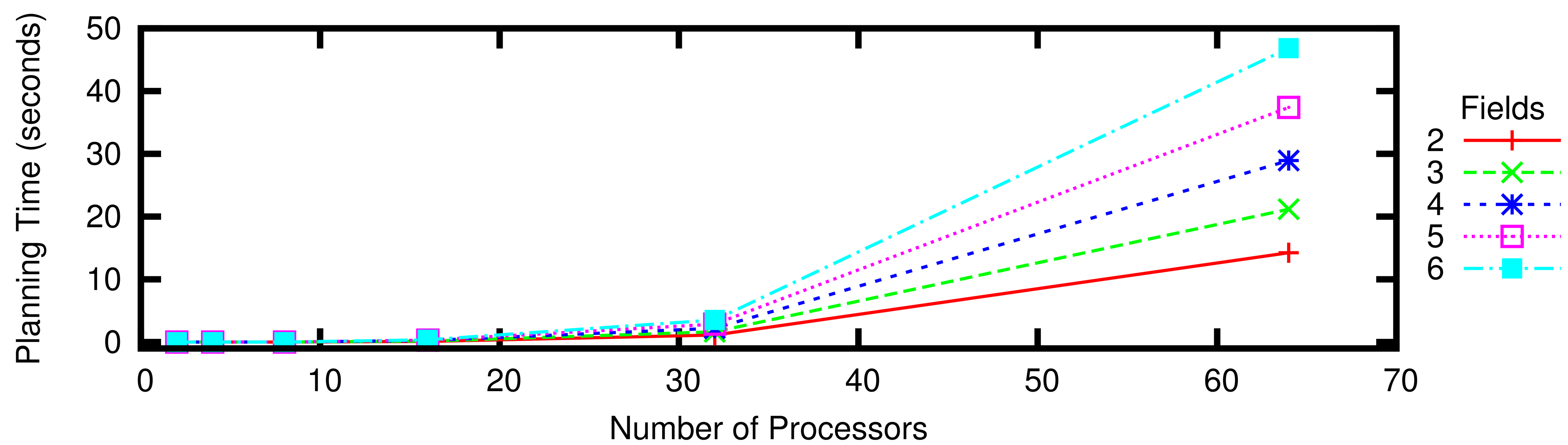
Agg(n_1, f_1, $\sigma_{hash(PK)=1}(T)$)
Agg(n_1, f_2, $\sigma_{hash(PK)=1}(T)$)
Agg(n_2, f_1, $\sigma_{hash(PK)=2}(T)$)
Agg(n_2, f_2, $\sigma_{hash(PK)=2}(T)$)
Agg(n_3, f_1, $\sigma_{hash(PK)=3}(T)$)
Agg(n_3, f_2, $\sigma_{hash(PK)=3}(T)$)
Agg(n_4, f_1, $\sigma_{hash(PK)=4}(T)$)
Agg(n_4, f_2, $\sigma_{hash(PK)=4}(T)$)
transmit(n_2, CNT($f_1, \sigma_{hash(PK)=2}(T)$), n_1)
transmit(n_3, CNT($f_1, \sigma_{hash(PK)=3}(T)$), n_1)
transmit(n_4, CNT($f_1, \sigma_{hash(PK)=4}(T)$), n_1)
merge(n_1, f_1)
transmit(n_1, CNT($f_2, \sigma_{hash(PK)=1}(T)$), n_2)
transmit(n_3, CNT($f_2, \sigma_{hash(PK)=3}(T)$), n_2)
transmit(n_4, CNT($f_2, \sigma_{hash(PK)=4}(T)$), n_2)
merge(n_2, f_2)

Query Execution Plan using `Dagg`

DAgg(n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$)
DAgg(n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$)
DAgg(n_3, f_1, f_2, $\sigma_{hash(PK)=3}(T)$)
DAgg(n_4, f_1, f_2, $\sigma_{hash(PK)=4}(T)$)
transmit(n_2, CNT($f_1, \sigma_{hash(PK)=2}(T)$), n_1)
transmit(n_3, CNT($f_1, \sigma_{hash(PK)=3}(T)$), n_1)
transmit(n_4, CNT($f_1, \sigma_{hash(PK)=4}(T)$), n_1)
merge(n_1, f_1)
transmit(n_1, CNT($f_2, \sigma_{hash(PK)=1}(T)$), n_2)
transmit(n_3, CNT($f_2, \sigma_{hash(PK)=3}(T)$), n_2)
transmit(n_4, CNT($f_2, \sigma_{hash(PK)=4}(T)$), n_2)
merge(n_2, f_2)

- Solved by GBFS using relaxed plan heuristic in Fast Downward

- Table below shows planning time for different values of F and N



- Solutions were optimal (although this is not guaranteed)

Acknowledgements

This work was supported by the Technion-Microsoft Electronic-Commerce Research Center.