SDA: Software-Defined Accelerator for general-purpose big data analysis system

Jian Ouyang(ouyangjian@baidu.com),
Wei Qi, Yong Wang,
Yichen Tu, Jing Wang, Bowen Jia

Baidu is beyond a search engine

- Search

- O2O/autonomous car/cloud/finance…

Outline

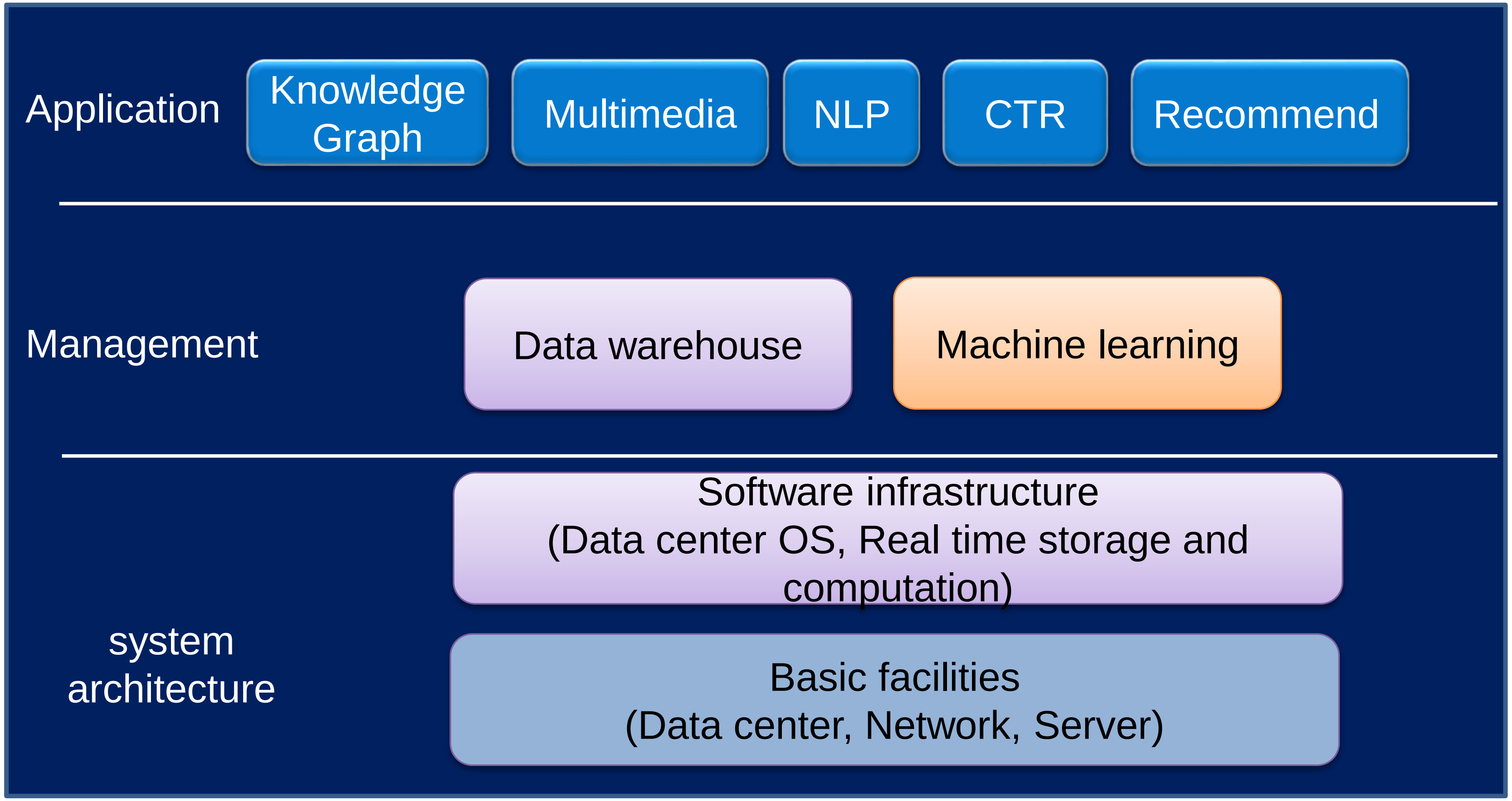
- The overview of big data and its system
- Motivations and challenge of accelerating big data processing
- Our solution : SDA for big data
  - Design goals
  - Design and implementation
  - Performance evaluation
- Conclusion

The overview of big data and its system

Total data:	~1EB
Processing data :	~100PB/day
Total web pages:	~1000 Billion
Web pages updated:	~10Billion/day
Requests:	~10Billion/day
Total logs :	~100PB
Logs updated:	~1PB/day

Data is big

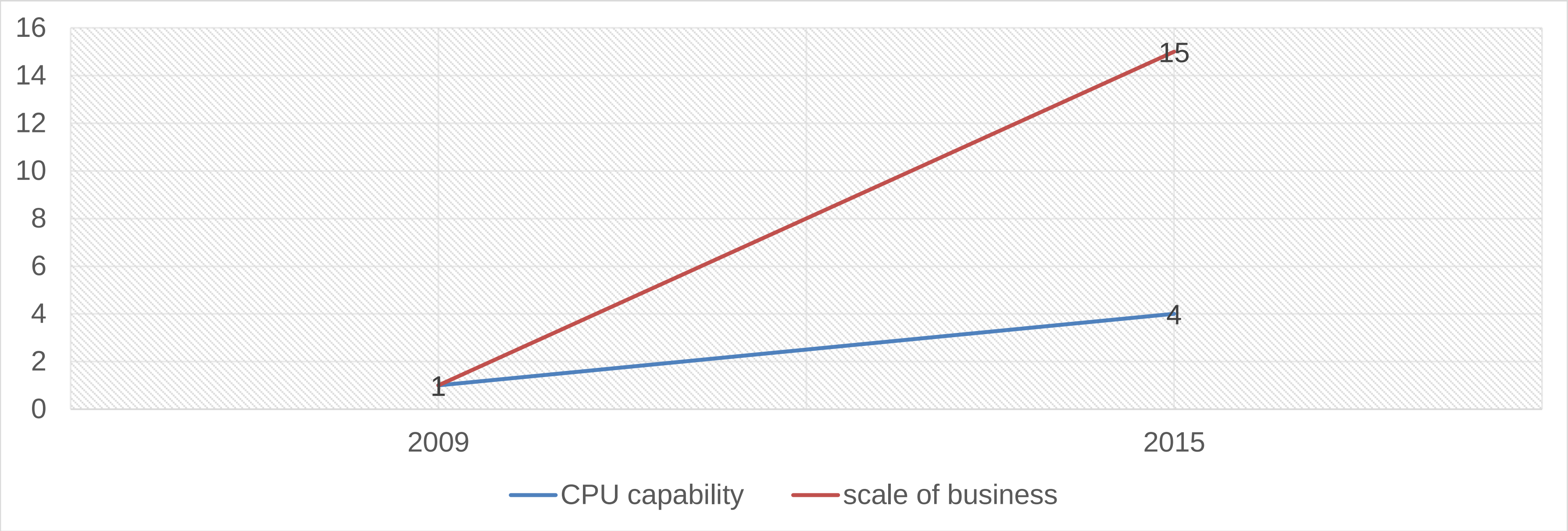
The overview of big data and its system



- Applications are diverse
- Systems are complex

Bai du 百度

Motivation of accelerating big data processing



- CPU typical configuration in data center
  - 2009, dual-core, ~3.0Ghz
  - 2015, 8-core, ~2.0Ghz
- Scale of business (Baidu revenue)
  - 2009, ~4.4B RMB
  - 2015,~66B RMB

- Computing = data scale * algorithm complexity
- Trends
  - Data scale increase rapidly
  - Algorithm complexity increase rapidly
  - Performance of CPU increase slowly
- Need to bridge the gap

Challenges of accelerating general-purpose big data processing

- Difficult to abstract the computing kernels
  - Diversities of big data applications
  - Variable computing type
- Difficult to be integrated into distributed system
  - Variable platforms and program models
    - MR
    - Spark
    - Streaming
    - User defined
  - Variable data type and storage format

Our solution : SDA – Software-Defined Accelerator

- Our observations
  - ~40% data analysis jobs written in SQL
  - Most others of data analysis jobs can be rewritten in SQL
  - Lots of popular SQL system
    - HIVE
    - Spark SQL
    - Impala…

- our solution
  - Software-Defined Accelerator for data analysis system

SDA – design goal

- Focus on general purpose data analysis system
  - SQL accelerator
    - Hiding the diversities of big data workloads

- Can be easily integrated into distributed system
  - For example, Spark SQL

SDA design – abstraction of SQL kernels

- Operations abstraction
  - Filter
  - Sort
  - Aggregate
  - Join
  - Group by

- Data type
  - Char/uchar/short/ushort
  - Int/uint/long/ulong
  - Float/double(time)
  - string

```sql
select
  i_item_id,
```

aggregate

```sql
  avg(ss_quantity) agg1,
  avg(ss_list_price) agg2,
from
```

join

```sql
  store_sales
  join date_dim on (ss_sold_date_sk = d_date_sk)
where
```

filter

```sql
  cd_gender = 'F'
  and d_year = 1998
  and ss_sold_date_sk between 2450815 and 2451179
```

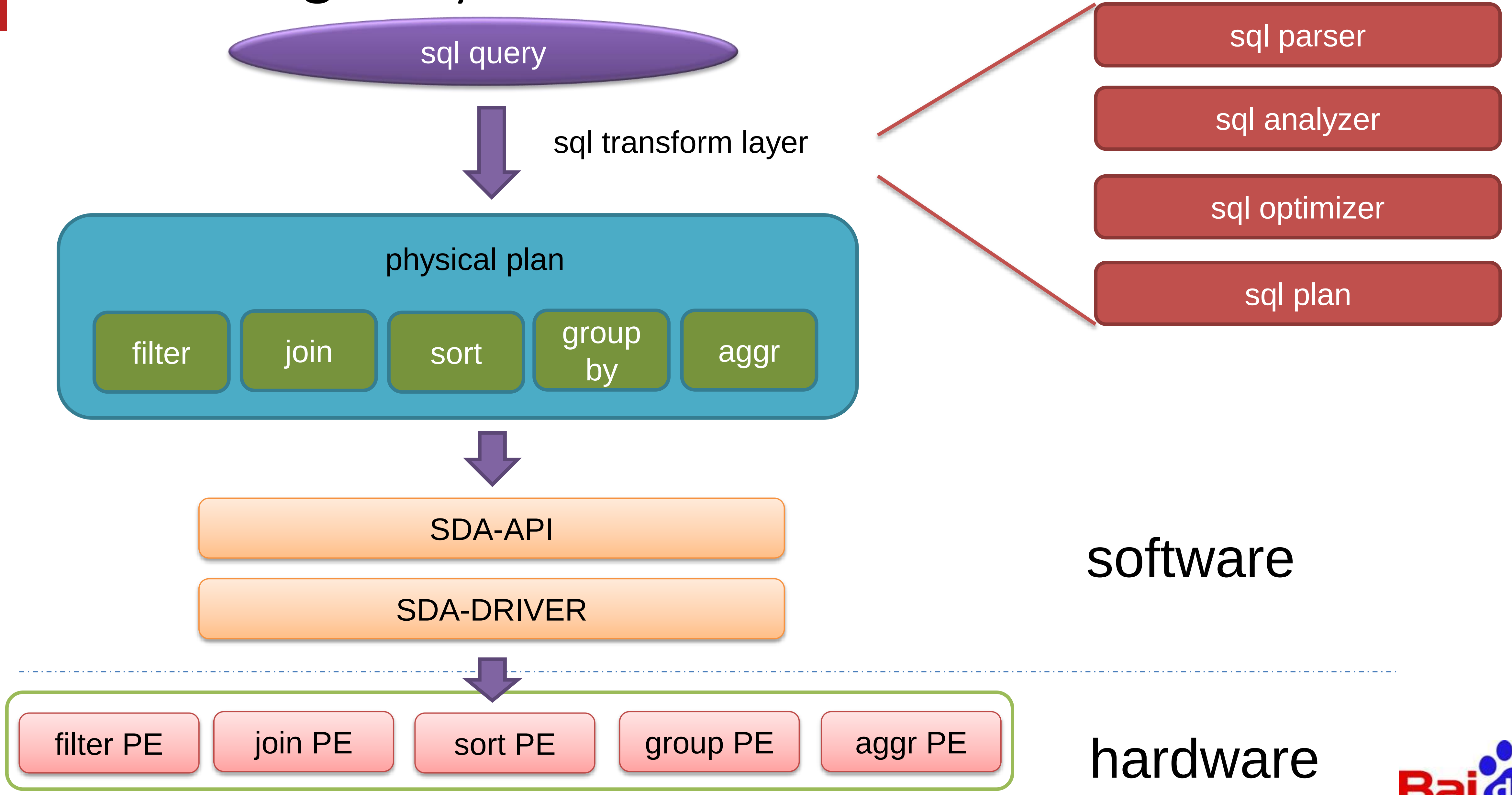
group by

```sql
group by
  i_item_id
```

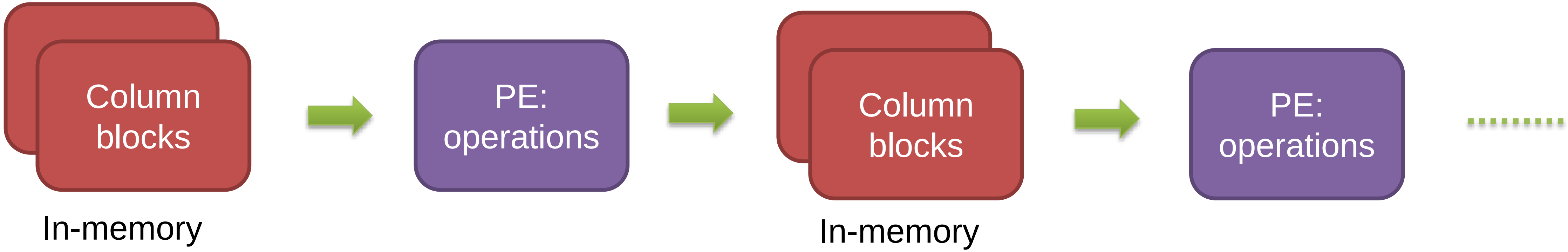
sort

```sql
order by
  i_item_id
```

SDA design - system



SDA design– data flow program model



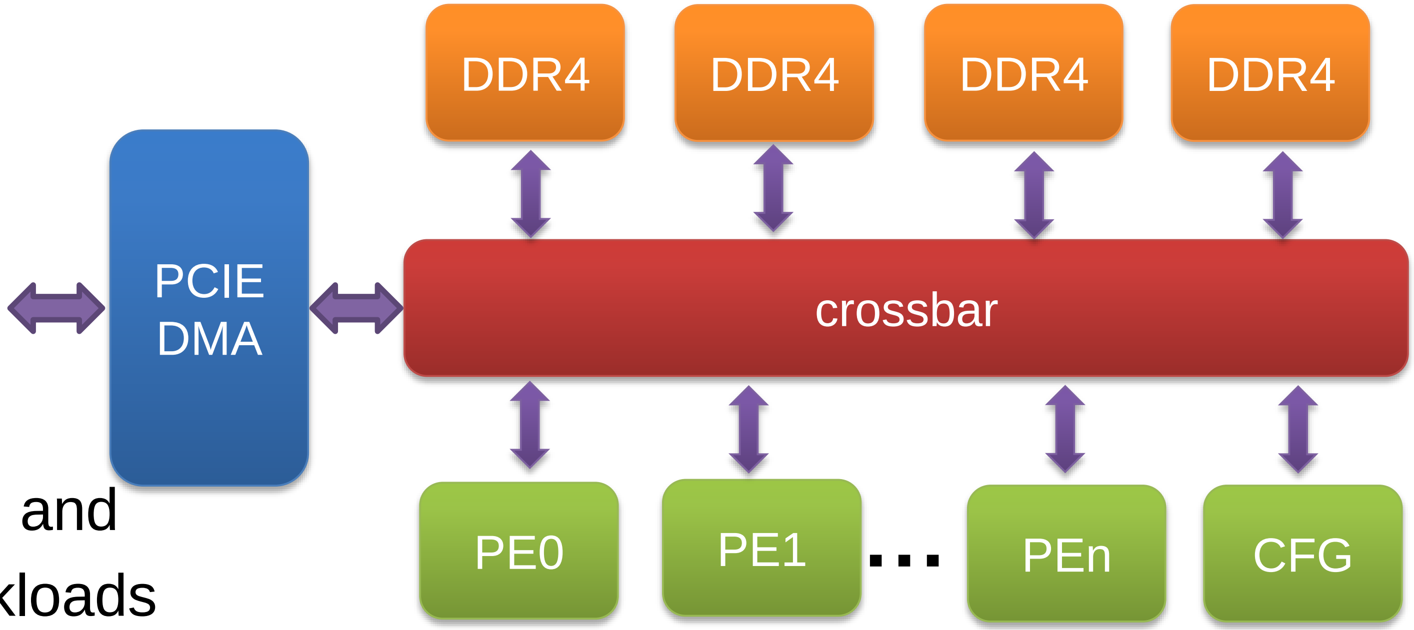
- Data blocks resident in on-board DDR memory
  - Column store
  - Reduce communication overhead

- Copy block to host
  - The operations which are not supported by SDA
  - Shuffle operation

Baidu 百度

SDA design– hardware

- PE(processing element)-based architecture
  - Scalable
  - Flexible



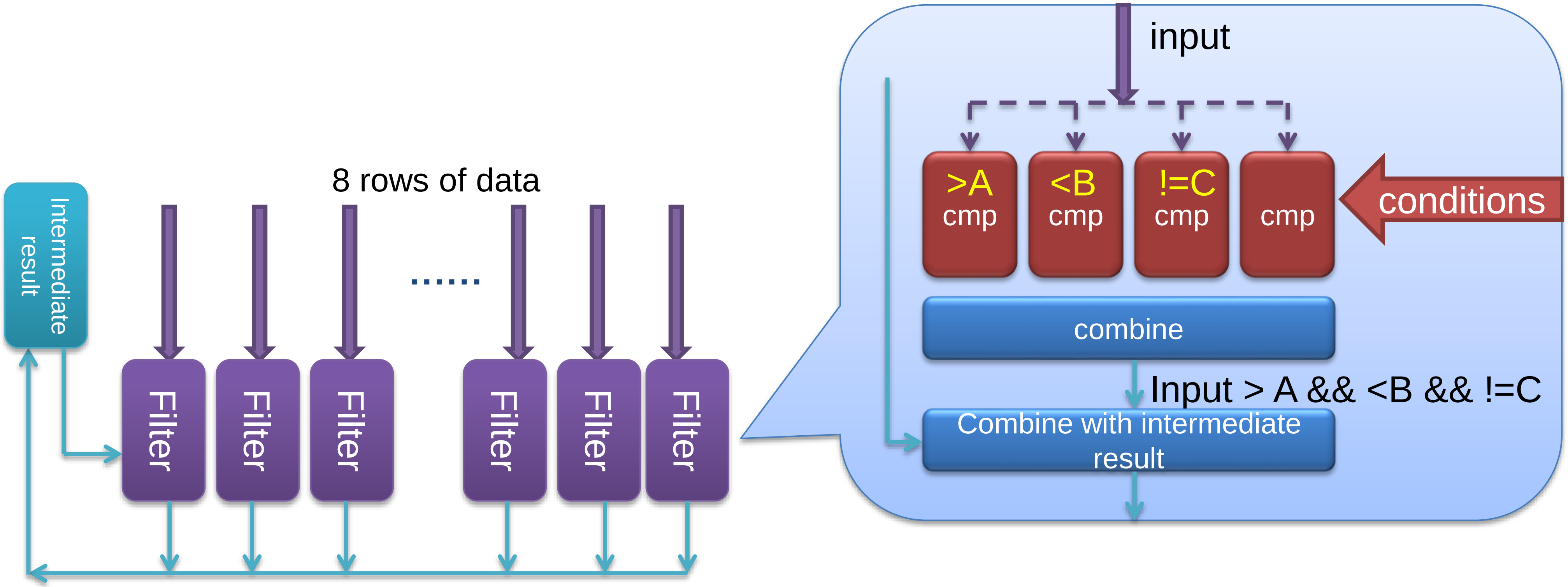
- Dynamic re-configurable
  - Resource on-demand
  - Dynamically configure the kind and Number of PEs according to workloads

- PE
  - Single operation: filter/sort/join…
  - Support variable data type

Baidu 百度

SDA design: filter PE micro-architecture

- Represent Filter conditions by postfix expression
- Highly parallel data path



SDA Implementation - hardware board

- Full-height, half-length
  - Feasibility for large-scale deployment
- Xilinx KU115 FPGA
- ~50W total power
- 4 x 72bit DDR4, 2400MHz
  - 8GB ~ 32GB capacity
  - ~76GB/s bandwidth with ECC
  - Big data applications are memory bandwidth bound
- PCI-e 3.0 x 8 lanes

SDA implementation – FPGA logic

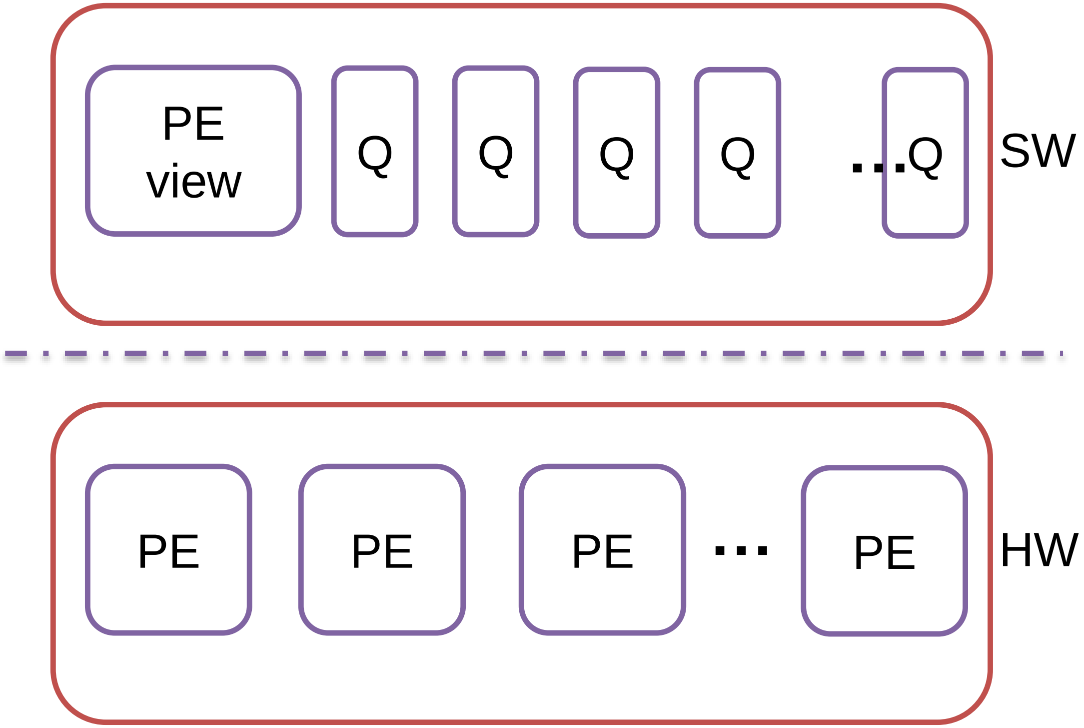
- Running at 300MHz

- RTL flow

Function	LUT	BRAM	DSP
Filter	16k (2.5%)	67 (3%)	3 (~0%)
Sort	85k (12.8%)	331(15%)	150(2.7%)
Aggregate	11k (1.6%)	56 (2.5%)	20 (~0%)
Join	15k (2.4%)	600(27%)	0
Group by	95k (14%)	380(17%)	170(3%)

SDA implementation – integration with distributed system

- Driver
  - Configure PEs according to workload
  - Allocate Queues for PE
  - Maintain the views of PE



- APIs
  - C++ library

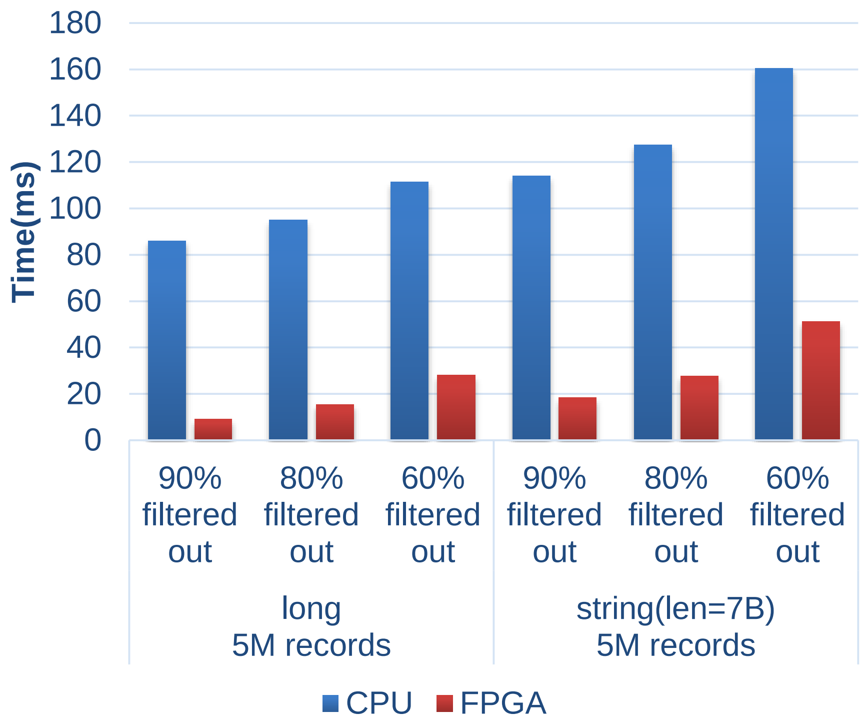
```
bool filter(int fd,
    __in const column* columns, uint8_t col_count,
    __out column* results, uint8_t ret_col_count,
    const op_info* expr, uint8_t op_count);

bool sort(int fd,
    __in const column* columns, uint8_t col_count, uint8_t j, bool descend,
    __out column* results, uint8_t ret_col_count);
```

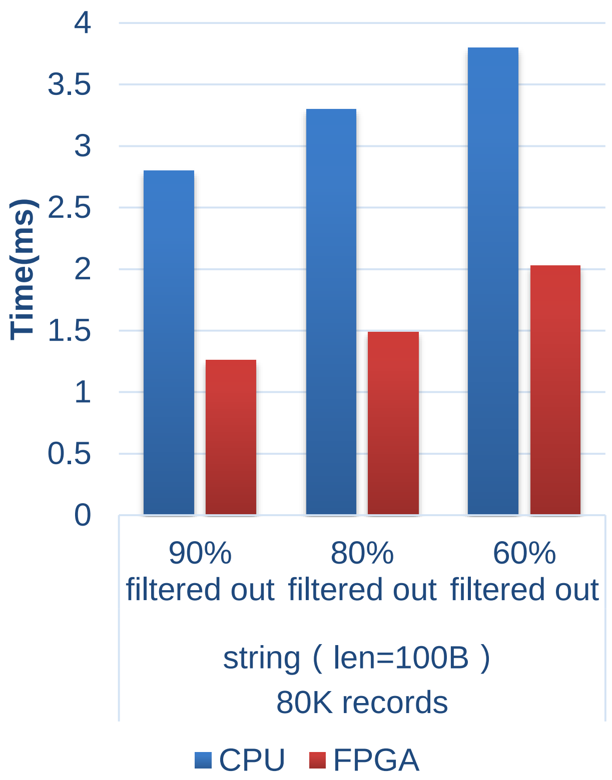
Evaluation

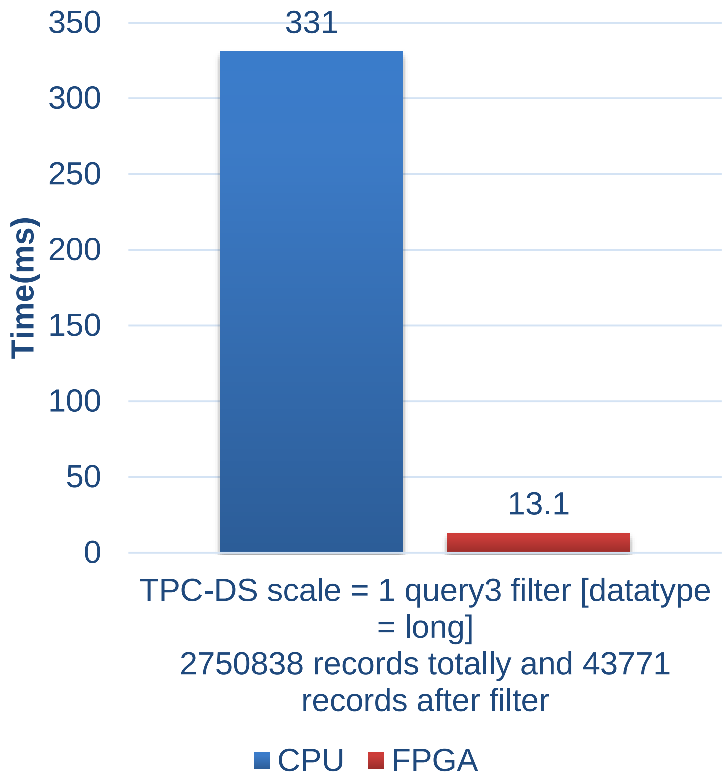
- Setup
  - Host
    - Intel E2620 x2, 2.0Ghz, 12 cores
    - 128GB memory
    - Linux
  - SDA
    - 5 PEs: filter, sort, aggregate, join, group by
    - 300MHz

Evaluation - filter

- Micro benchmark
  - At most 10x than dedicated C++ comparator
- Real case: TPC-DS query3
  - 25x faster than general C++ comparator



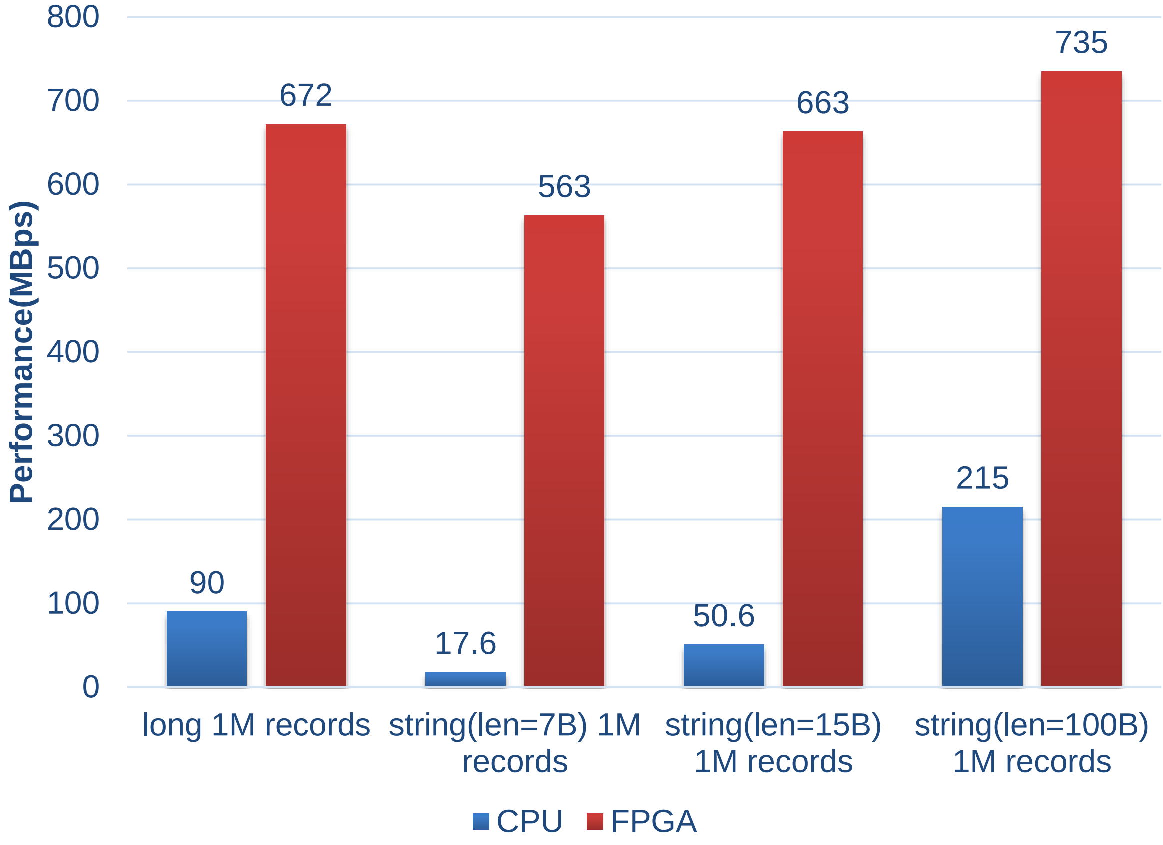




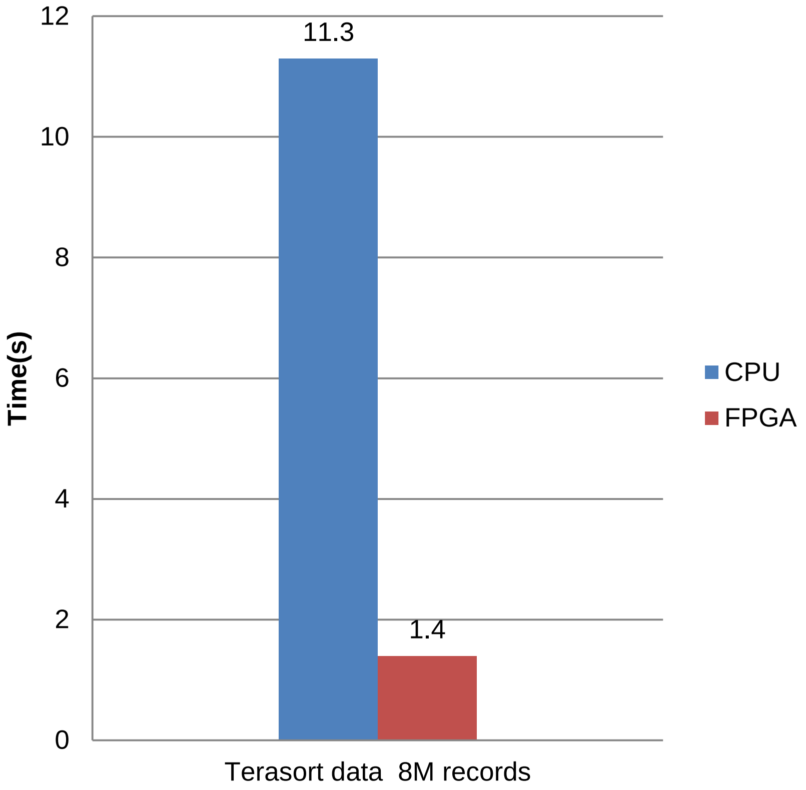
Baidu百度

Evaluation-sort

- Micro benchmark
  - At most 33x

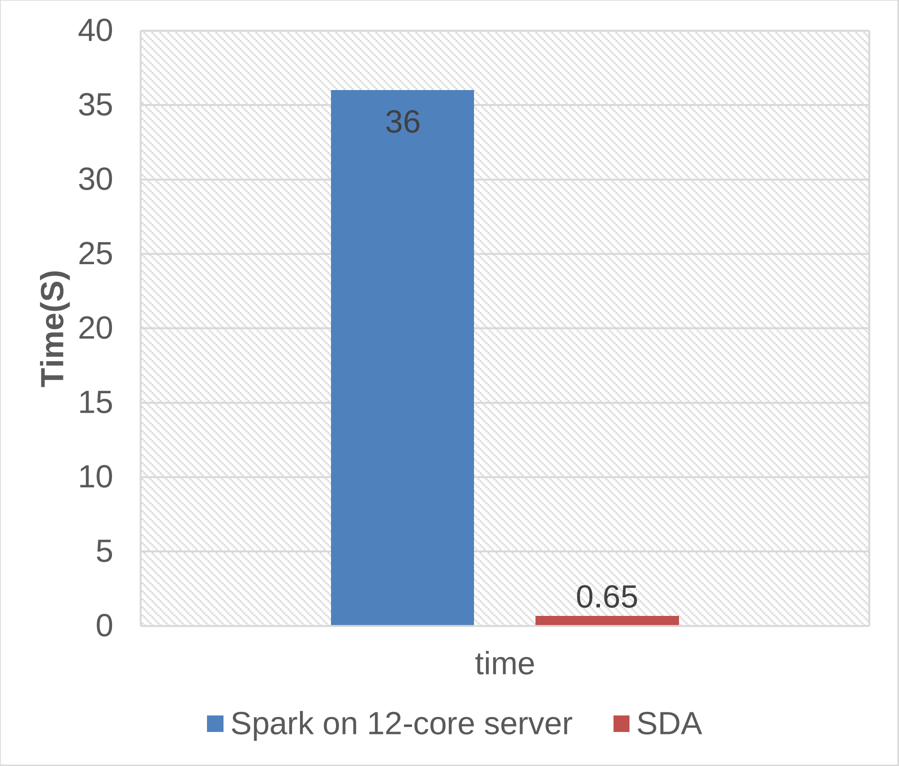


- Terasort
  - 8x



Evaluation - real case query

- TPC-DS scale = 10， query3
- Execution time
  - 55x




```sql
select
  dt.d_year,
  item.i_brand_id brand_id,
  item.i_brand brand,
  sum(ss_ext_sales_price) sum_agg
from
  store_sales
  join item on (store_sales.ss_item_sk = item.i_item_sk)
  join date_dim dt on (dt.d_date_sk = store_sales.ss_sold_date_sk)
where
  item.i_manufact_id = 436
  and dt.d_moy = 12
  and (ss_sold_date_sk between 2451149 and 2451179
    or ss_sold_date_sk between 2451514 and 2451544
    or ss_sold_date_sk between 2451880 and 2451910
    or ss_sold_date_sk between 2452245 and 2452275
    or ss_sold_date_sk between 2452610 and 2452640)
group by
  d_year,
  item.i_brand,
  item.i_brand_id
order by
  d_year,
  sum_agg desc,
  brand_id
limit 100
```

Related work

- Sorting
  - GTX Titan : ~500M int64 on 1M inputs (https://nvlabs.github.io/moderngpu/mergesort.html)
  - SDA: 84M int64 of one PE, 336 M of 4 PEs
  - SDA is ~4x power efficiency than GPU

- GPU/FPGA accelerator for SQL kernels
  - Lots of papers and some startups
    - Only support a subset of data type and operation
  - No work on whole TPC-DS query
    - GPU is not good at complex data type, such as variable length string
    - Most keys of real workload are variable length string, such as city, people and item

- ASIC accelerator for SQL operations
  - Q100, implemented by 32nm ASIC http://arcade.cs.columbia.edu/q100-asplos14.pdf
  - The accelerations kernels of SDA are similar to Q100, such as join, sort and aggregation.
  - From applications perspective, Q100 is designed for data base, but SDA targets to data analysis(SparkSQL, HIVE)
  - From design perspective, Q100 explore design space in-depth to balance power and performance. SDA is limited by current FPGA platform, such as memory bandwidth, logic resource and max frequency. But thanks to FPGA, SDA can be dynamically reconfigured according to workload demands.
  - Q100 has NOC for extremely efficient data pipeline. SDA only shares data via off-chip DRAM among kernels for flexibility.
  - SDA is implemented by really hardware. And we design APIs for integrating SDA to real software system. the evaluations are in real system.

- Advantages of SDA
  - Support all data type of TPC-DS
  - General-purpose, support most operations of TPC-DS
  - High performance benefited from
    - Random memory access
    - Data locality
    - Customized pipeline and ALU

Conclusion

- Present general-purpose big data accelerator
  - Design for general-purpose big data analysis system, such as SparkSQL, HIVE
  - Abstract SQL operations
  - Propose the SDA hardware and software architecture
  - Implement SDA hardware by FPGA
- SDA is also designed for distributed system
  - Data flow program model
  - Resource on-demand
- Demonstrate the feasibility of SDA for big data and AI
  - SDA for AI on Hotchips 2014