Want extreme performance at scale? Do distributed the RIGHT way!

Valentin Kulichenko
Apache Ignite Committer
GridGain Solutions Architect
Where Is The Challenge?
Data Affinity
Where Entry Goes?

```
put (key, value)
```

Ignite Node 1

Ignite Node 2
Caches and Partitions

Cache

Partition 1
- K1, V1
- K2, V2
- K3, V3
- K4, V4

Partition 2
- K5, V5
- K6, V6
- K7, V7
- K8, V8
- K9, V9
Partitions Distribution

Ignite Node 1

0  2  4  6  8

10 12 14

Ignite Node 2

1  3  5  7  9

11 13 15
Affinity Function

Key → Partition → Server Node

ON-DISK
Where Entry Goes?

put (key, value)

Ignite Node 1
0 2 4

Ignite Node 2
1 3 5
Co-located Processing
1. Initial Request
2. Fetch data from remote nodes
3. Process entire data-set

1. Initial Request
2. Co-located processing with data
3. Reduce multiple results in one
Use Case: Account Balance Update

```java
class Account {
    String firstName;
    String lastName;
    String address;
    
    double balance;
}

Account account = cache.get(123);
account.balance -= 100;

cache.put(123, account);
```
Use Case: Account Balance Update

cache.invoke(123, new EntryProcessor<Integer, Account, Object>() {
    @Override public Object process(MutableEntry<Integer, Account> entry, Object... args) {
        Account account = entry.getValue();

        account.balance -= 100;

        entry.setValue(account);

        return null;
    }
});
Co-located Data
Use Case: Payment Transaction Authorization

```java
class Transaction {
    int accountId;
    String storeName;
    double amount;
}
```

For each new transaction:
- Find all transactions for the account ID
- Go through the list, calculate authorization variables
- If transaction is authorized, add it to the list
Affinity Key

Key → Affinity Key → Partition → Server Node

Account ID
class TransactionKey {
    int transactionId;
    @AffinityKeyMapped
    int accountId;
}

ignite.compute().affinityRun("transactions", 123, () -> { ... });
Co-location and SQL: Indexing
Let’s Run a SQL Query!

```
SELECT AVG(amount) FROM Transaction WHERE accountId = ?
```
Executing SQL: Full Scan

- 1/3x latency
- 3x capacity
But What If We Use Index?
Indexed Search Complexity

\[ \log 1\_000\_000 \approx 20 \]

\[ \log 333\_333 \approx 18 \]

\[ \log 333\_333 \approx 18 \]

\[ \log 333\_333 \approx 18 \]
Executing SQL: Indexed Search

- ~same latency
- ~same capacity
Let’s Co-locate

```
SELECT AVG(amount) FROM Transaction WHERE accountId = ?
```
Executing SQL: Indexed Search With Co-location

- same latency
- 3x capacity

1 node

3 nodes
Co-location and SQL: Joins
Random Distribution

Ignite Node

Canada
Toronto
Mumbai
Calgary

India
Montreal
Ottawa
New Delhi
Non-Collocated Joins

1. Initial Query
   
   ```sql
   SELECT ct.name, count(c.name)
   FROM Country as ct
   JOIN City as c ON c.countryCode = ct.code
   WHERE ct.name IN ('Canada', 'India') GROUP BY (ct.name);
   ```

2. Query execution (local + remote data)

3. Potential data movement

4. Reduce multiple results in one
Affinity Collocation

- Key (countryId = 5)
- Key (cityId = 10, countryId = 5)
- Key (cityId = 11, countryId = 9)
Collocated Distribution

- **Canada**: Toronto, Montreal, Ottawa, Calgary
- **India**: Mumbai, New Delhi
Collocated Joins

1. Initial Query
2. Query execution over local data
3. Reduce multiple results in one

```
SELECT ct.name, count(c.name)
FROM Country as ct
JOIN City as c ON c.countryCode = ct.code
WHERE ct.name IN ('Canada', 'India') GROUP BY (ct.name);
```
Any Questions?

Thank you for joining us. Follow the conversation.
http://ignite.apache.org

@vkulichenko
#apacheignite