Fetch data in parallel

Improving Data Reading Performance

How slow is the JDBC of the database?

【Testing Example】

From 30 million rows and 8 columns of customer table (text size 4.9 GB), fetch data for testing from Oracle and MySQL respectively.

Testing result:（Unit：second）

	First Time	Second Time	Rows per second
Oracle	293	281	106000
MySQL	518	381	79000

Hardware environment： Two Intel2670 CPU，2.6GHz，16 core in total，Memory 64G，SSD hard disk

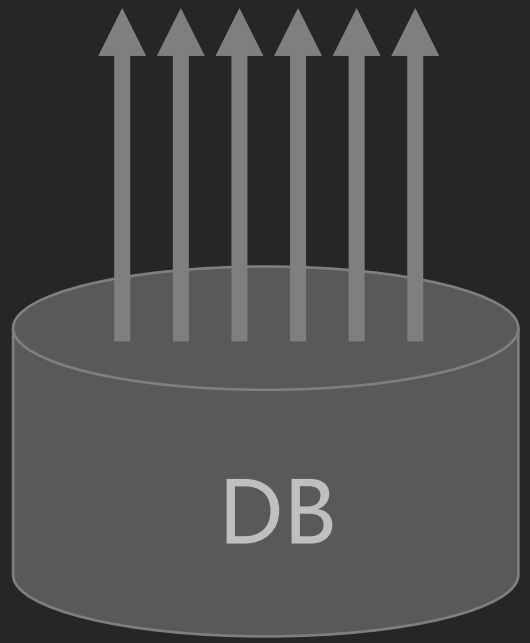
It takes 5 minutes to read a 30 million row！！！ OMG

How to speed up?

Accelerate reading efficiency by parallel fetching and utilize multi-CPU capability.

However, JAVA is too difficult to implement parallel programs (have to consider troublesome transactions such as resource sharing conflicts)

Is there a simple way?



Parallel for single table

1. How to segment?
   a. Segmentation with index
   b. Segmentation without index
2. External storage Case

Setting parallel number

1. Not more than CPU core number
2. Far more than CPU core number

Parallel for multi-table

Fetch data in parallel and Join calculation

Using Index Field to Segment

Segment equally: Data segmentation is needed before single table can be retrieved in parallel to ensure that the data of each segment is relatively equal.

Using index can speed up the positioning of segment data and fast implement parallel data fectching.

Example

Parallel reading of order table data for a specified period.
(Index field: orderID)

Code（1）- Accurate calculation of data range

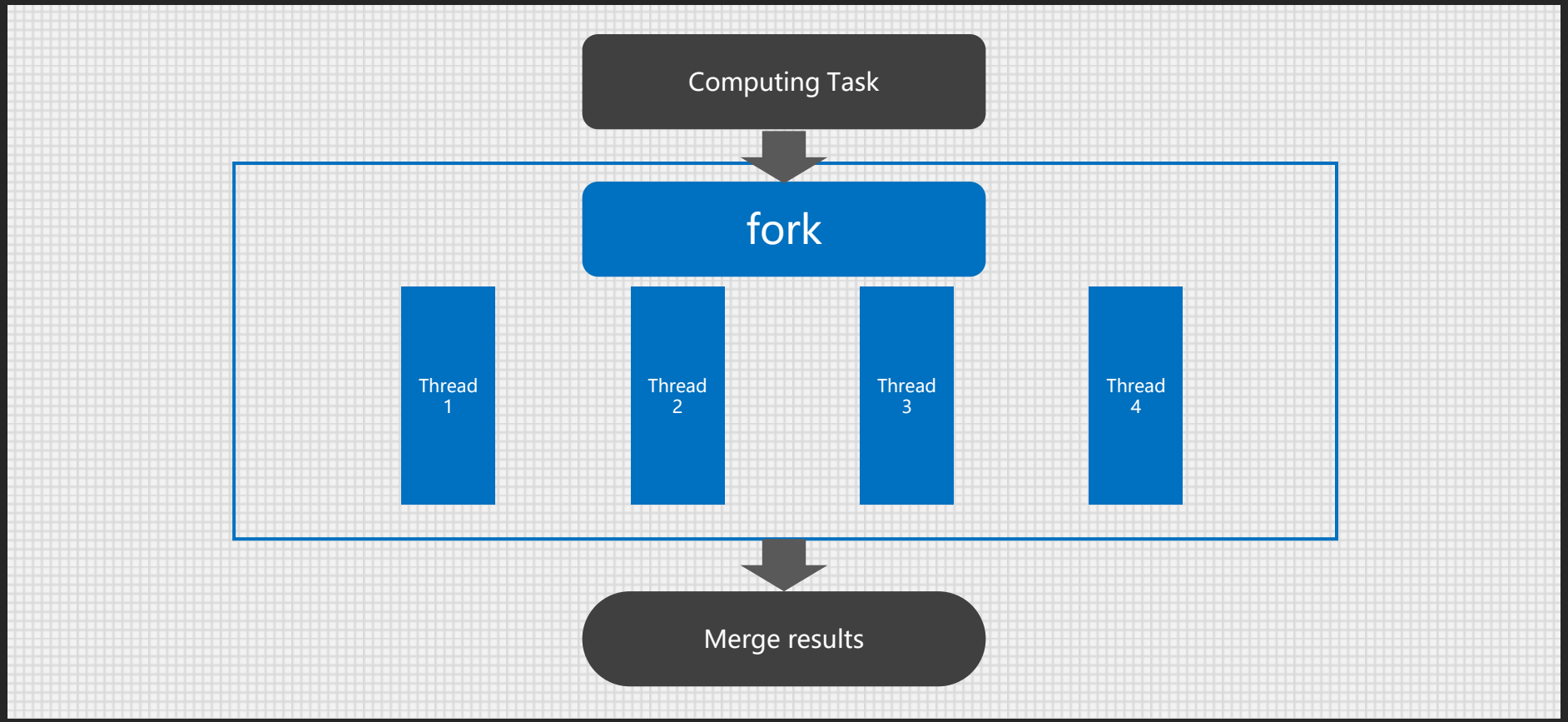
	A	B	C
1	=connect("db")		
2	=A1.query("select min(orderID) minID,max(orderID) maxID from order where orderDate>=? and orderDate<=?",begin,end)	=b=A2.minID	=e=A2.maxID
3	=p=4	/Parallel number	=range(b,e+1,p)
4	=C3.to(,p)	/Initial value of segment parameter	
5	=C3.to(2,)	/End value of segment parameter	
6	fork A4,A5	=connect("db")	
7		=B7.query@x("select * from order where orderID>=? and orderID<? and orderDate>=? and orderDate<=?",A6(1),A6(2),begin,end)	
8	=A6.conj()	/Merge query result	

Code（2）- Effective use of index

	A	B	C
1	=connect("db")		
2	=A1.query("select min(orderID) minID,max(orderID) maxID from order)	=b=A2.minID	=e=A2.maxID
3	=p=4	/Parallel number	=range(b,e+1,p)
4	=C3.to(,p)	/Initial value of segment parameter	
5	=C3.to(2,)	/End value of segment parameter	
6	fork A4,A5	=connect("db")	
7		=B7.query@x("select * from order where orderID>=? and orderID<? and orderID>=? and orderID<?",A6(1),A6(2),begin,end)	
8	=A6.conj()	/Merge query result	

About fork

In esProc, multiple threads can be started by fork statement to implement parallel computing, and esProc also provides a variety of merge functions to facilitate the merging of parallel results.



Note

It should be noted that the database connection must be established in parallel threads so that it can be used separately for multiple threads. If a common connection is used, the acceleration can’ t be achieved because the database will automatically change multiple requests on the same connection to serial execution. Therefore, parallel fetches can be used to improve performance only when the database is not overburdened and there are enough connections available.

ATTENTION

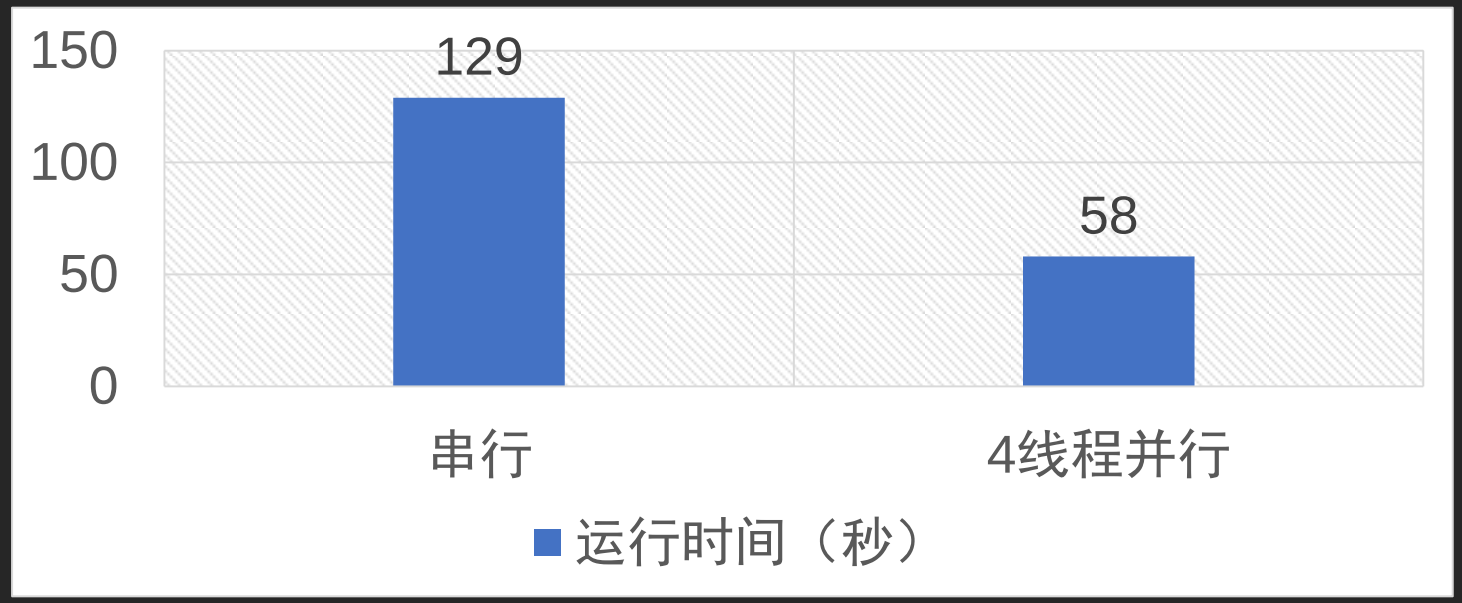
Performance comparison

Testing example： Read order data for a specified period based on the order table.

Testing environment（PC）：

CPU：i5-6200U@2.30GHz
RAM：8GB
OS：win10-64bit
DB：oracle12c
Order table record number：5 million
Read data range：3 years data（8 years data in total）

Testing result：



Non-indexed field segmentation

If the database burden is not heavy, it can also be segmented based on non-indexed field (such as date). Compared with JDBC data fetching time, the time to traverse the database table multiple times is not very large.

Benefits: There is no need to query database first to determine the start and end of each segment.

Example	Parallel reading of order table data for a specified period.

Code

	A	B	C
1	=connect("db")		
2	=range(begin,end+1,4)		
3	=p=4	/Parallel number	
4	=A2.to(,p)	/Initial value of segment parameter	
5	=A2.to(2,)	/End value of segment parameter	
6	fork A4,A5	=connect("db")	
7		=B6.query@x("select * from order where orderID>=? and orderID<?",A6(1),A6(2))	
8	=A6.conj()	/Merge query result	

External storage case

Sometimes a statement (a table) has a large amount of data. Parallel sub-task can not fit into memory after segmentation. We can query by cursor and then merge in this case.

Example

Parallel reading of order table data for a specified period.

(Data is so large that it can’ t fit into memory after segmentation.)

Code

	A	B	C
1	=connect("db")		
2	=A1.query("select min(orderID) minID,max(orderID) maxID from order where orderID>=? and orderID<=?",begin,end)	=b=A2.minID	=e=A2.maxID
3	=p=4	/Parallel number	=range(b,e+1,p)
4	=C3.to(,p)	/Initial value of segment parameter	
5	=C3.to(2,)	/End value of segment parameter	
6	=A4.(connect(“db”).cursor@x(“select * from order where orderID>=? and orderID<? and orderID>=? and orderID<=?”,~,A5(#),begin,end))		
7	=A6.mcursor()	/Merge query result	
8	=file(“\usr\order.txt").export@t(A7)	/Write file based on cursor	

Note

Parallel query based on external storage cursor is very similar to the in-memory mode. When the memory resource is tight, it can reduce the memory consumption by external storage computing.

When cursor merges, because the running speed of each thread can not guarantee regularity, the data export sequence based on multi-thread is uncontrollable, and this method can not be used when the data sequence is required.

Pay attention to the requirement of result orderliness

Setting Parallel Number

Normally, the parallel number of a parallel program is recommended not to exceed the number of CPU cores （≤CPU core number）

Because more tasks do not increase parallelism, and can avoid additional time overhead caused by thread switching in CPU.

Case 1:

Parallel number≤CPU core number

Setting Parallel Number

Setting the number of tasks far larger than the CPU core number (multiple CPU core number), so that multi-CPU load can be dynamically balanced, and for some calculations the segmentation can also be simplified.

Case 2:

Parallel number far larger than CPU core number

Example

Parallel reading of order table data in one year.

The number of threads can be set to 12 (months) when querying only one year's data, thus simplifying segmentation.

	A	B	C
1	fork to(1,12)	=connect("demo")	
2		=B1.query@x("select * from order where month(orderDate)=? and orderID>=? and orderID<=?",A1,begin,end)	
3	=A1.conj()	/Merge query result	

Note

esProc provides a dynamic balancing mechanism for multi-threaded tasks. When the number of tasks is larger than the number of parallel setting, esProc automatically assigns the next task to the thread that finished current calculation. It can ensure that one thread runs several small tasks, while the other thread runs only a smaller number of large tasks to achieve overall balance, so that the amount of data does not have to be distributed equally.

Flexibility

Parallel for multi-table

In some multiple SQL query scenarios, such as multiple datasets of reports, data can still be retrieved by executing multiple statements simultaneously in parallel.

order
orderID
orderDate
customerID
eID
…

customer
customerID
customerName
area
…

employee
eID
eName
…

orderDetail
orderID
productID
price
quantity
…

product
productID
productName
type
…

Example

Read 5 tables data in parallel and complete join.

Code

	A	B	C
1	=connect("db")		
2	="select * from order where orderID>=date('"/begin/"') and orderID<=date('"/end/"')"		
3	select orderID,productID,price, quantity from orderDetail		
4	select customerID,cunstomerName from customer		
5	select eID,eName from employee		
6	select productID,productName from product		
7	fork [A2:A6]	=connect("db")	
8		=B7.query@x(A7)	
9	=od=A7(1)	=detail=A7(2)	
10	=cus=A7(3)	=emp=A7(4)	=prod=A7(5)
11	>od.switch(customerID,cus:customerID;eID,emp:eID)		
12	=detail.switch(orderID,order:orderID;productID,prod:productID)		
13	=A12.new(orderID.customerID.customerName:cusName,orderID.orderID:orderID,orderID.eID.eName:empName,productID.productName:prod,price, quantity)		

Higher performance based on files

If the data is moved out of the database and put into the file system, the performance will be better. esProc provides an efficient data storage format - set files and group tables.

Data fetching performance comparison

Summary

The significance of esProc multi-threading parallelism lies in its simplicity and low cost. Compared with JAVA complex multi-threaded programming, esProc can be as simple as several lines of codes. Compared with the database cluster scheme, the cost esProc is more controllable. Moreover, even if the database cluster is deployed, esProc can still be used to accelerate the data fetching of individual database node in the cluster.

Simple and low cost