CockroachDB支持在一个事务中并行执行独立的INSERT、UPDATE、UPSERT、DELETE语句。并行执行语句有助于减少聚合延迟并提高性能。

为什么使用并行语句执行

传统SQL引擎在一个事务中顺序执行SQL语句，服务端执行语句计算并将每个语句的执行结果返回给客户端。只有在客户端接收到语句返回值后，它才会发送下一个要执行的SQL语句。

在传统单节点的数据库中，语句在一台机器上执行，所以执行过程不会带来通信延迟。然而，对于CockroachDB这类分布式复制的数据库来说，语句的执行可以跨越多个节点，节点之间的协调将带来通信延迟。顺序执行SQL语句会带来更高的累积延迟。

然而对于并行语句执行，同个事务中的多个SQL语句同时被执行，从而减少总延迟。

并行语句执行的工作原理

在以下场景中，让我们来理解顺序和并行执行是如何工作的：

* 假设我们想更新一个社交网络应用中用户的last name，favorite movie和favorite song
* 数据库有三个表需要被更新：users, favorite\_movies和 favorite\_songs。

传统事务更新用户的信息如下：

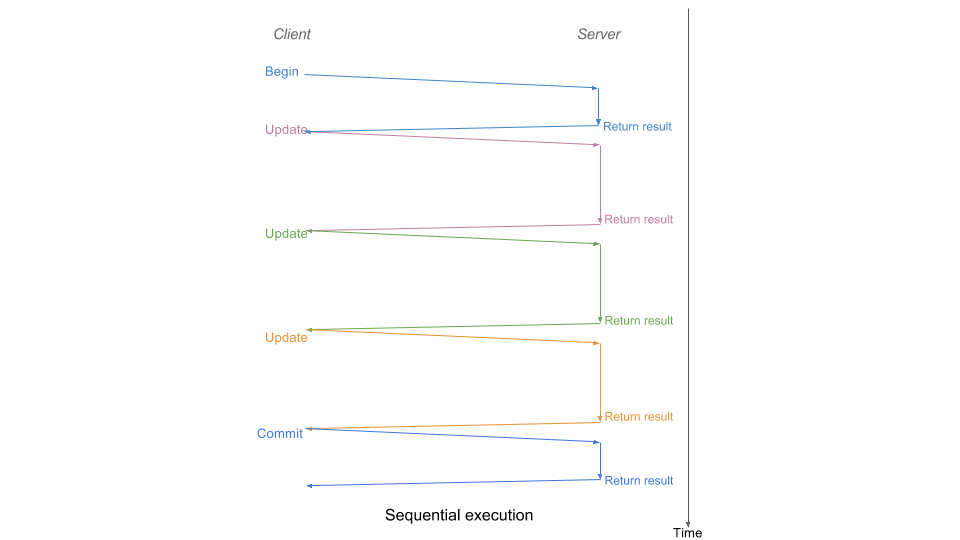
> **BEGIN**;

> **UPDATE** users **SET** last\_name = 'Smith' **WHERE** id = 1;

> **UPDATE** favorite\_movies **SET** movies = 'The Matrix' **WHERE** user\_id = 1;

> **UPDATE** favorite\_songs **SET** songs = 'All this time' **WHERE** user\_id = 1;

> **COMMIT**;

顺序执行一个事务中的SQL语句时，服务器在执行完语句后将发送返回值。客户端只有在接收到上一个语句的返回值后才会发送下一个需要执行的SQL语句。这通常成为“conversational API“，如以下概念图所示： 

我们的示例场景中的SQL语句可以并行执行，因为它们彼此独立。为了并行执行语句，客户端不需要等待先前的SQL语句的返回结果就可以发送下一条需要执行的语句。在CockroachDB中，使用在SQL语句附加RETURNING NOTHING子句，服务端会立即发送确认，而不是等待执行完语句再将返回值发送给客户端。客户端接收到确认后，发送下一条可以执行的语句。这使得CockroachDB可以并行执行语句。这些语句是并行执行的，直到CockroachDB遇到**barrier语句**。barrier语句是任何没有RETURNING NOTHING子句的语句。服务端将会顺序执行barrier语句。

在示例场景中，事务将会执行如下：

> **BEGIN**;

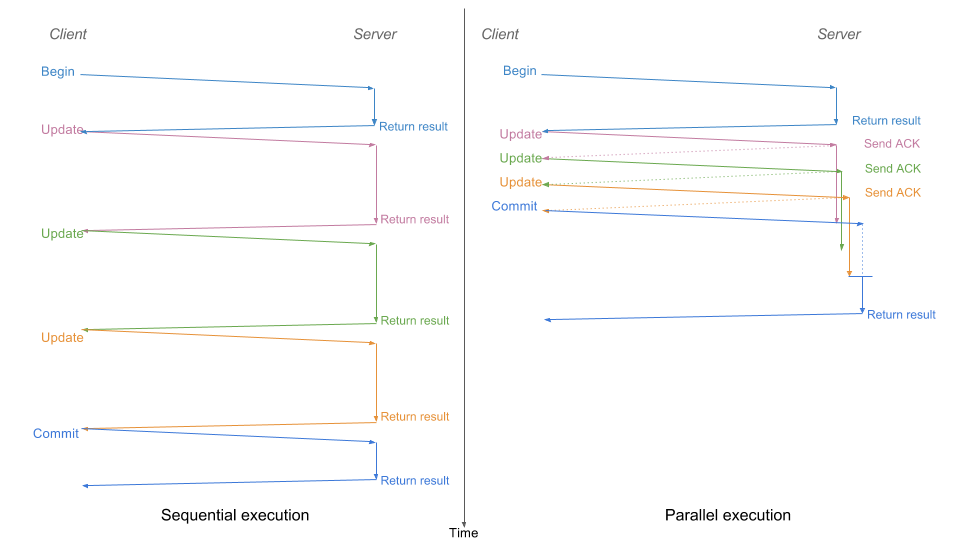
> **UPDATE** users **SET** last\_name = 'Smith' **WHERE** id = 1 RETURNING NOTHING;

> **UPDATE** favorite\_movies **SET** movies = 'The Matrix' **WHERE** user\_id = 1 RETURNING NOTHING;

> **UPDATE** favorite\_songs **SET** songs = 'All this time' **WHERE** user\_id = 1 RETURNING NOTHING;

> **COMMIT**;

在这个场景中，因为事务中的UPDATE语句是彼此独立的，它们可以并行执行而不会影响结果。COMMIT语句是barrier语句，会顺序执行。只有在其前面的所有并行语句都已完成执行后，才会执行barrier语句。

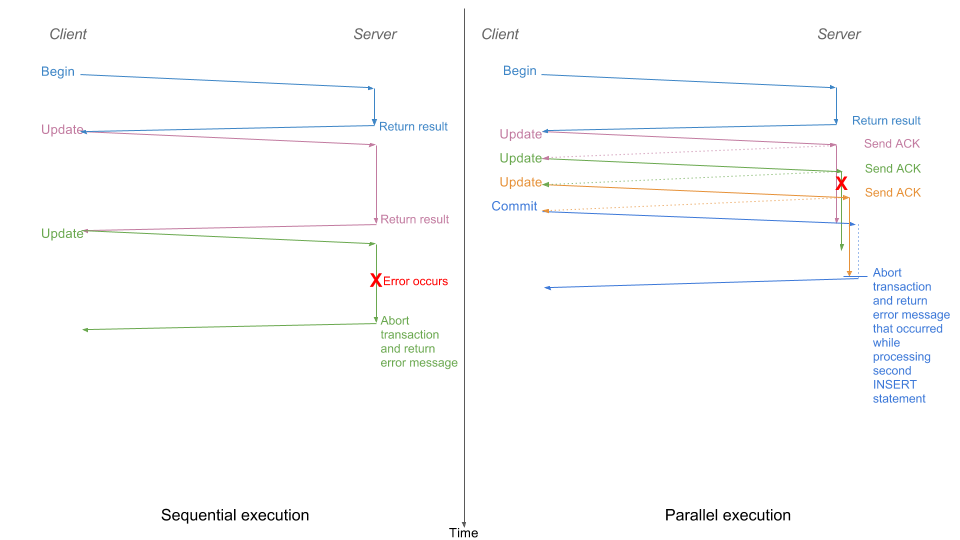
下图展示了事务是如何按顺序和并行执行的，并且展示了并行执行语句是如何减少总延迟的： 

在barrier语句的执行中感知延迟

如前所述，服务端只有在其前面的所有并行语句都已完成执行后，才会执行barrier语句。因此看起来似乎barrier语句执行时间更长，不过它是在等待并行语句执行。即使这样，并行执行语句后，顺序执行barrier语句所需的总时间也应该小于顺序执行所有语句所需的时间。

在上图提到的，服务器在执行COMMIT前会等待所有UPDATE语句执行完成，所以看起来COMMIT花了更长时间执行，但实际上它实在等待UPDATE语句执行完成。

错误信息不匹配

在顺序执行时发生错误，事务将会被中止并且发送错误信息给客户端。然而，在并行执行时，信息不会在遇到错误时发送，而是在下一个barrier语句之后发送。这将会造成客户端收到的错误信息和正在执行的sql语句不匹配。下图解释了这种情况： 

在依赖语句后附加RETURNING NOTHING函数

如果两个连续的语句不是相互独立的，并且在语句中添加了RETURNING NOTHING 子句，CockroachDB检测依赖关系并顺序执行语句。这意味着你可以在SQL语句中使用RETURNING NOTHING子句，而不必担心它们的依赖性。

修改我们的示例场景，假设我们想在社交网络应用中创造一个新的用户，我们需要为用户的last name、favorite movie, and favorite song创建对应条目，并将它插入到三张表：users, favorite\_movies, and favorite\_songs。事务如下所示：

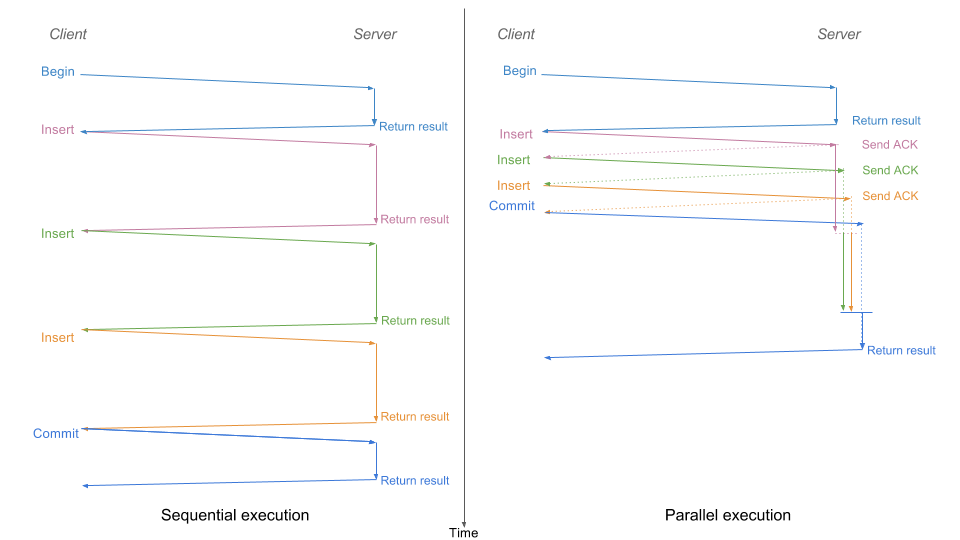
> **BEGIN**;

> **INSERT** **INTO** users **VALUES** last\_name = 'Pavlo' **WHERE** id = 2 RETURNING NOTHING;

> **INSERT** **INTO** favorite\_movies **VALUES** movies = 'Godfather' **WHERE** user\_id = 2 RETURNING NOTHING;

> **INSERT** **INTO** facvorite\_songs **VALUES** songs = 'Remember' **WHERE** user\_id = 2 RETURNING NOTHING;

> **COMMIT**;

在这种情况，第二和第三个INSERT语句依赖于第一个INSERT语句，因为movies和songs都和users表有外键约束。所以尽管我们在第一个语句后添加了 RETURNING NOTHING 子句，CockroachDB还是会依次执行第一条语句。在第一条语句执行完成后，第二和第三条INSERT语句会并行执行。以下概念图展示了事务是如何顺序和并行执行的： 

什么时候使用并行语句执行

一个事务中的SQL语句如果是相互独立的则可以并行执行。CockroachDB认为如果一个事务中的SQL语句调换顺序执行而不会影响执行结果，则可以认为是相互独立的。

举个例子，以下语句由于重排序后执行也不影响执行结果，因此被认为是相互独立的：

> **INSERT** **INTO** a **VALUES** (100);

> **INSERT** **INTO** b **VALUES** (100);

> **INSERT** **INTO** a **VALUES** (100);

> **INSERT** **INTO** a **VALUES** (200);

而以下的语句对则是有依赖关系的，因为重排序后执行将会影响结果：

> **UPDATE** a **SET** b = 2 **WHERE** y = 1;

> **UPDATE** a **SET** b = 3 **WHERE** y = 1;

> **UPDATE** a **SET** y = true **WHERE** y = false;

> **UPDATE** a **SET** y = false **WHERE** y = true;

CockroachDB中的并行语句执行和PostgreSQL中的并行查询执行是不一样的。对于PostgreSQL，并行查询执行是“创建多个查询进程，划分单个SQL语句的工作负载并并行执行它们“；而CockroachDB’s的并行语句执行，一个独立的SQL语句是不会被分到多个进程执行的， 相反，单个事务中的多个独立SQL语句是并行执行的。

See Also

* [INSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/INSERT/)
* [UPDATE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPDATE/)
* [DELETE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DELETE/)
* [UPSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPSERT/)