Group 12:

Pragmatic Projects - Web-scale Data Management

/ Flask - CockroachDB

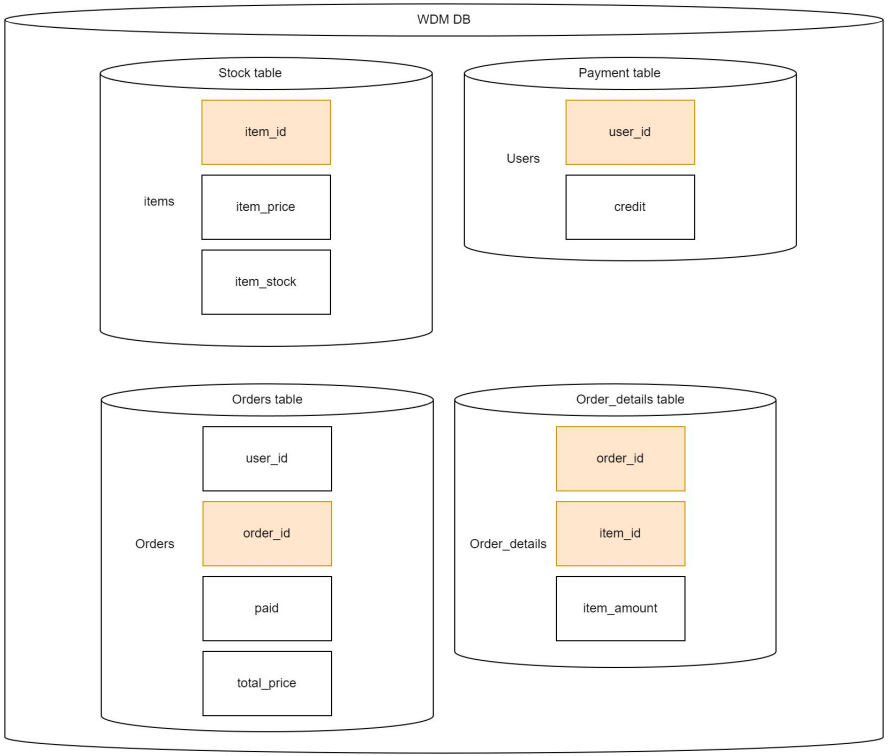
Yufan Tang(5701503), Zihan Wang(4840348)
Gefei Zhu(5651727), Zhiqiang Lei(5073812)

TU Delft

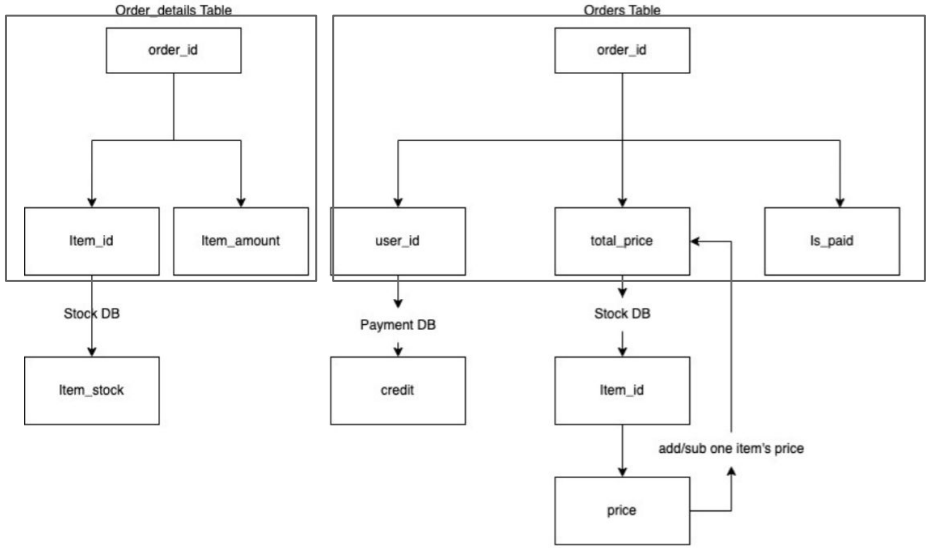
Database Design

CockroachDB

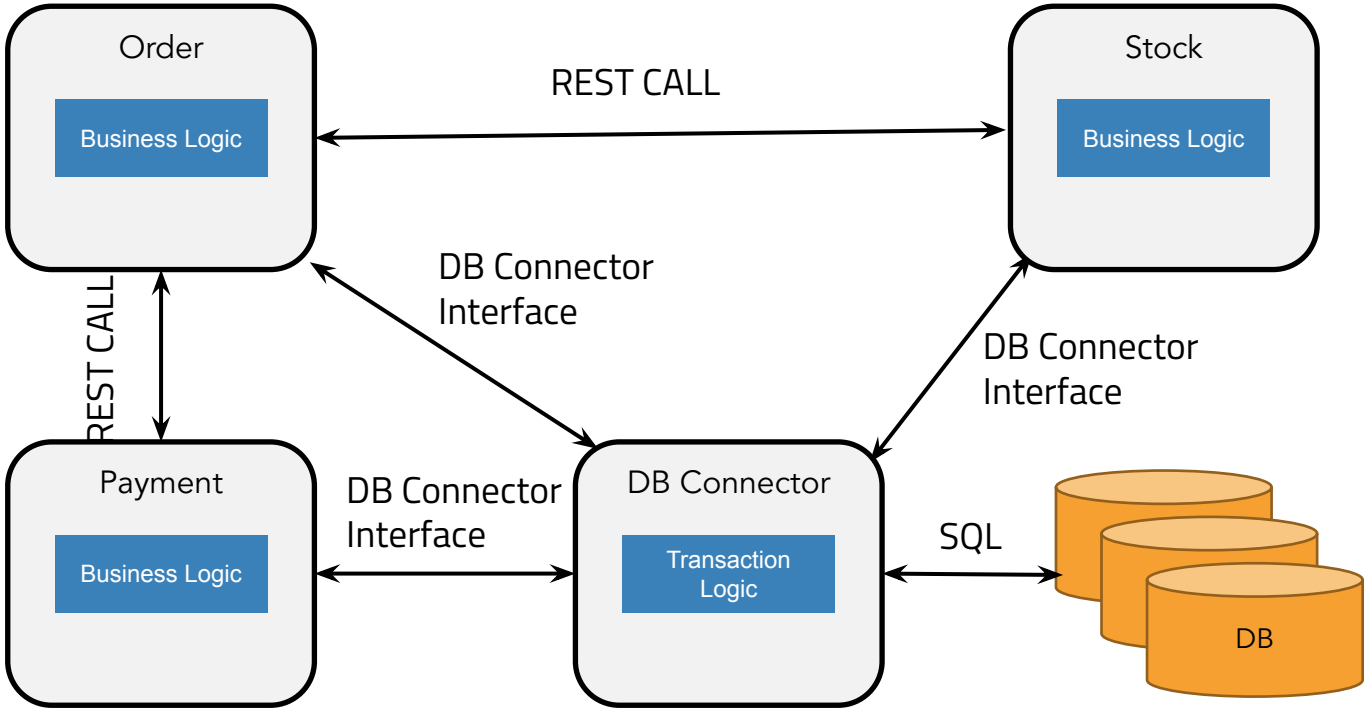
Overview



Order & Order_details example

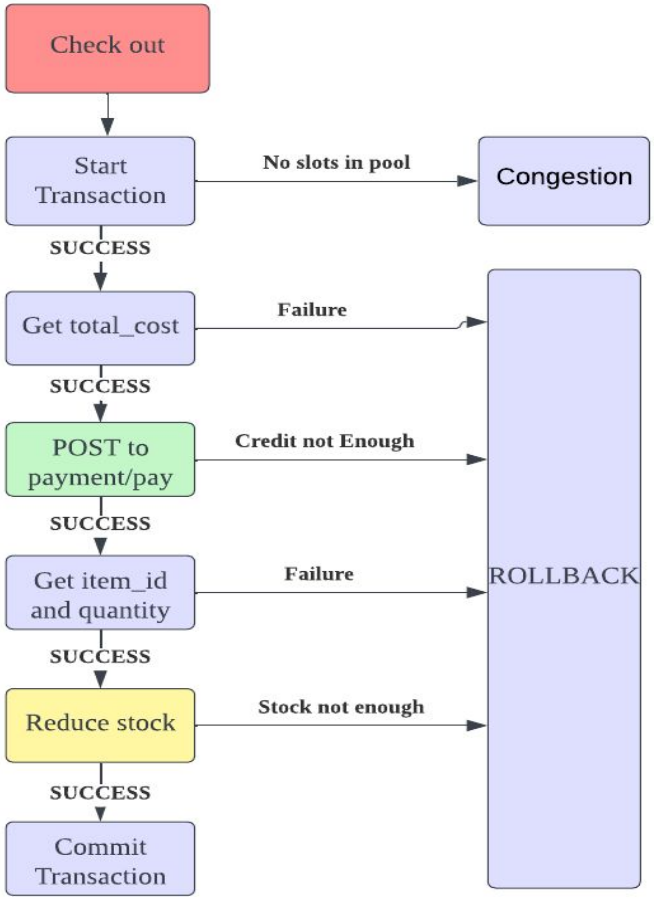


System Architecture Design



Transaction Design

- For requests involving single service:
  - Send SQL to DB directly
- For requests involving multiple services:
  - Start a transaction
  - Commit when all requirements are met
  - Otherwise rollback



Scalability

1. K8s auto-scale
2. Manual adjustment of memory in K8s(limited hardware resource)
3. Use connection pool of database

Performance & Availability & Fault tolerance

1. Load balancing in ingress service of K8s
2. Automatic restart of failed microservices in Kubernetes
3. Replica Service for Order
4. Failure recovery in CockroachDB ensures strong consistency across replicas (Raft Consensus protocol)

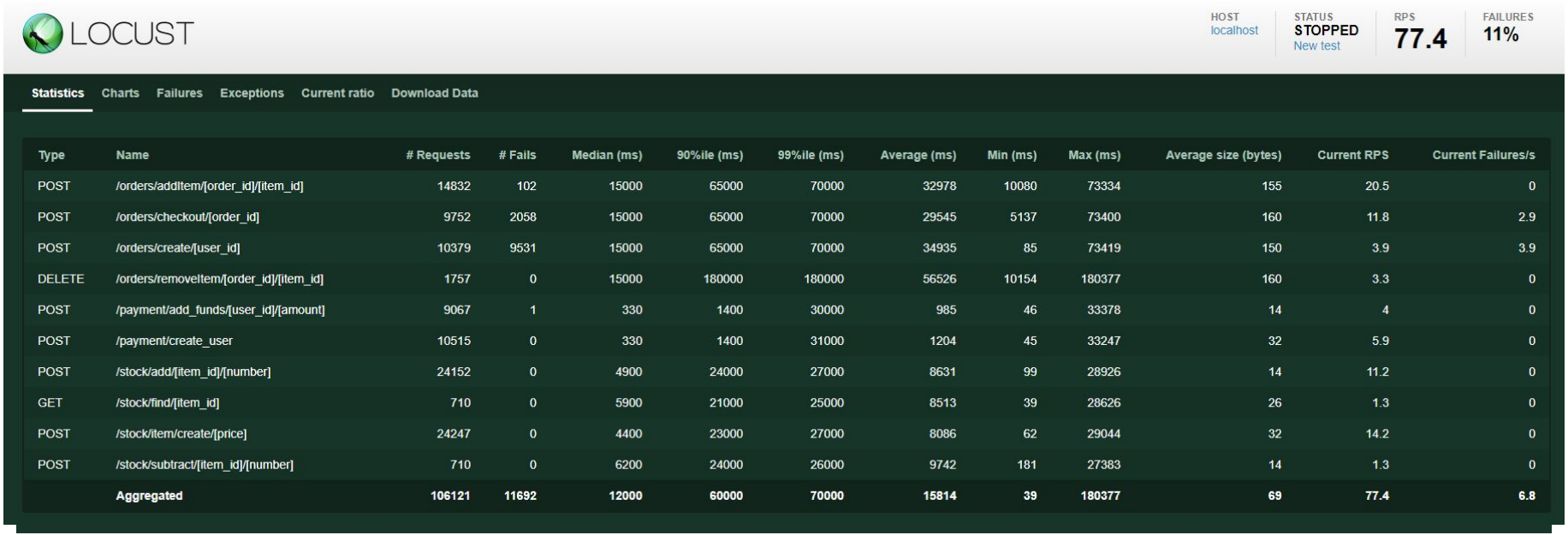
Consistency

1. We implemented 0 inconsistency
2. CockroachDB transaction prevents dirty write/read (isolation transaction)
3. Executing rollback when errors occur during execution.

```
INFO - 08:46:24 - populate - Items created
INFO - 08:46:24 - populate - Creating users ...
INFO - 08:48:13 - populate - Users created
INFO - 08:48:13 - Consistency test - Databases populated
INFO - 08:48:13 - Consistency test - Starting the load test...
INFO - 08:48:13 - stress - Creating orders...
INFO - 08:49:25 - stress - Orders created ...
INFO - 08:49:25 - stress - Running concurrent checkouts...
INFO - 08:50:19 - stress - Concurrent checkouts finished...
INFO - 08:50:19 - Consistency test - Load test completed
INFO - 08:50:19 - Consistency test - Starting the consistency evaluation...
INFO - 08:50:19 - verify - Stock service inconsistencies in the logs: 0
INFO - 08:50:26 - verify - Stock service inconsistencies in the database: 0
INFO - 08:50:26 - verify - Payment service inconsistencies in the logs: 0
INFO - 08:50:26 - verify - Payment service inconsistencies in the database: 0.0
INFO - 08:50:26 - Consistency test - Consistency evaluation completed
```

Stress test result

With 1200 users and 100 spawn rate we achieved the following result:

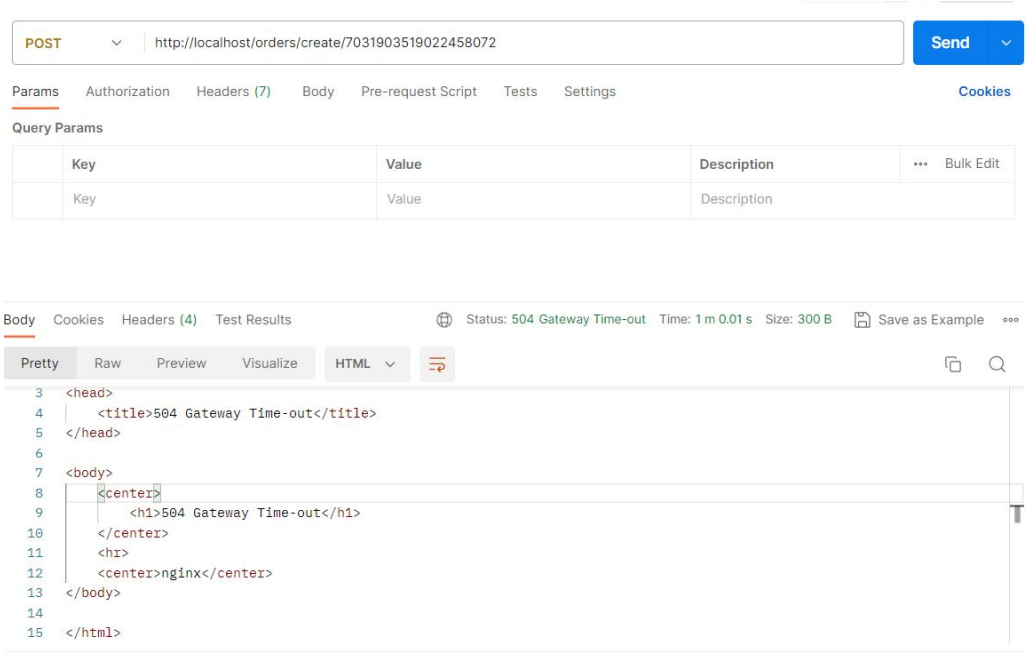


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/orders/addItem/[order_id]/[item_id]	14832	102	15000	65000	70000	32978	10080	73334	155	20.5	0
POST	/orders/checkout/[order_id]	9752	2058	15000	65000	70000	29545	5137	73400	160	11.8	2.9
POST	/orders/create/[user_id]	10379	9531	15000	65000	70000	34935	85	73419	150	3.9	3.9
DELETE	/orders/removeItem/[order_id]/[item_id]	1757	0	15000	180000	180000	56526	10154	180377	160	3.3	0
POST	/payment/add_funds/[user_id]/[amount]	9067	1	330	1400	30000	985	46	33378	14	4	0
POST	/payment/create_user	10515	0	330	1400	31000	1204	45	33247	32	5.9	0
POST	/stock/add/[item_id]/[number]	24152	0	4900	24000	27000	8631	99	28926	14	11.2	0
GET	/stock/find/[item_id]	710	0	5900	21000	25000	8513	39	28626	26	1.3	0
POST	/stock/item/create/[price]	24247	0	4400	23000	27000	8086	62	29044	32	14.2	0
POST	/stock/subtract/[item_id]/[number]	710	0	6200	24000	26000	9742	181	27383	14	1.3	0
	Aggregated	**106121**	**11692**	**12000**	**60000**	**70000**	**15814**	**39**	**180377**	**69**	**77.4**	**6.8**

Stress test result

Endpoint “orders/create/[item_id]” fails a lot due to high traffic

More errors occur with time going by.



- Add more replicas to order-service in k8s
- Increase the memory of order-service in k8s

Summary & Future work

1. Implements absolute consistency
2. Scalability of services(K8s Scaling)
3. Scalability of SQL queries(ConnectionPool)
4. Use replicas to ensure availability

1. Figuring out why create orders suffer the most from busy traffic
2. Improving the scalability while keeping the same consistency
3. Using another distributed database to see if there can be any improvement.
4. Play with connection pool configuration and more advanced pools