Group 4

Pragmatic Project

Web-scale data management

Aaron van Diepen

Thomas Eckhardt

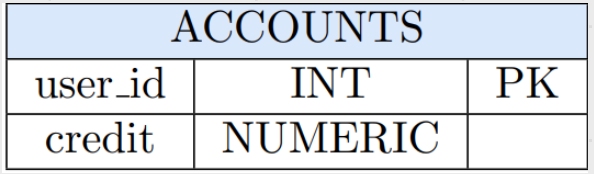
Justin Oosterbaan

Madelon Stol

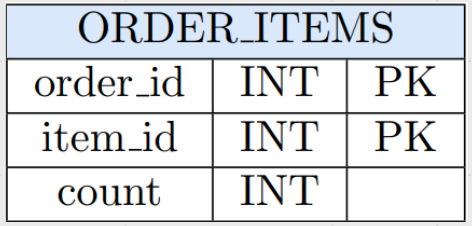
Jasper Teunissen

Technologies used

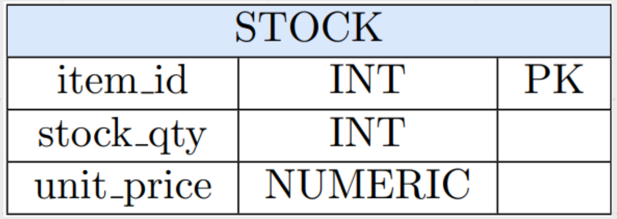
Database design



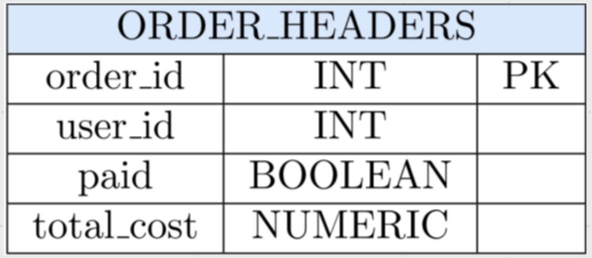
ACCOUNTS		
user_id	INT	PK
credit	NUMERIC	



ORDER_ITEMS		
order_id	INT	PK
item_id	INT	PK
count	INT	

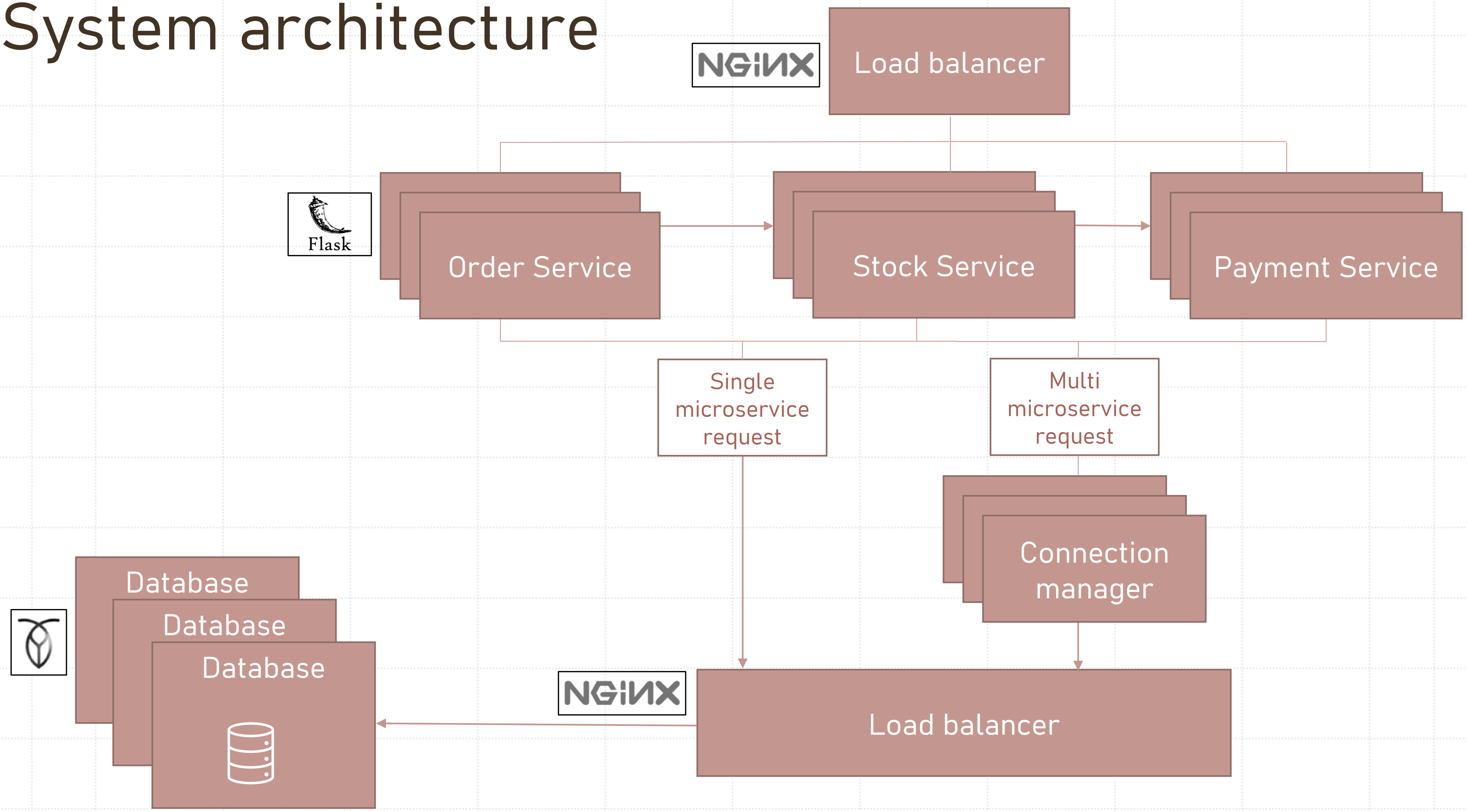


STOCK		
item_id	INT	PK
stock_qty	INT	
unit_price	NUMERIC	



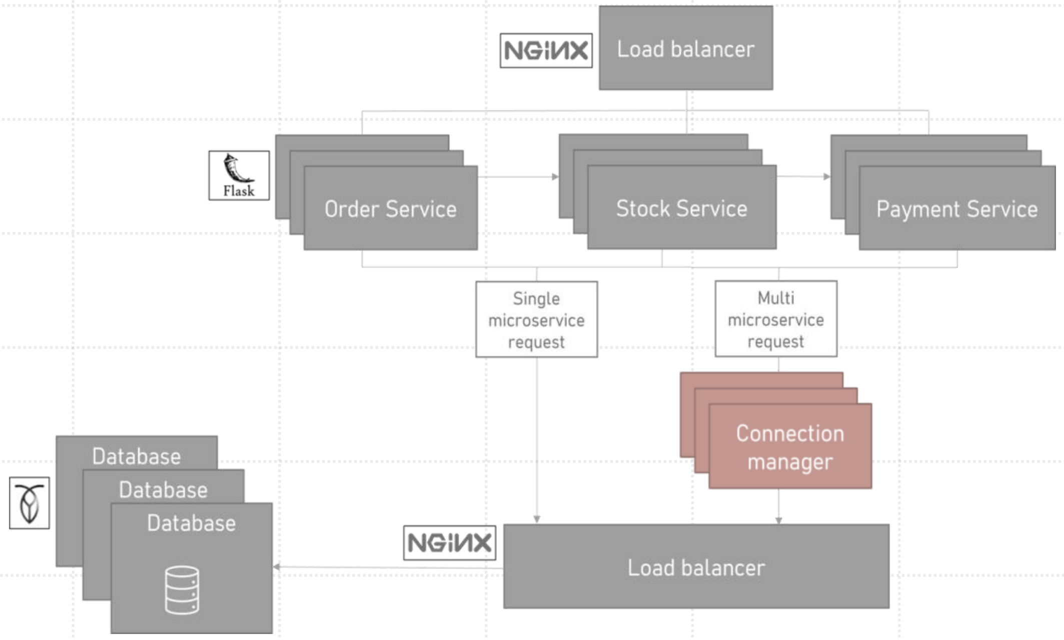
ORDER_HEADERS		
order_id	INT	PK
user_id	INT	
paid	BOOLEAN	
total_cost	NUMERIC	

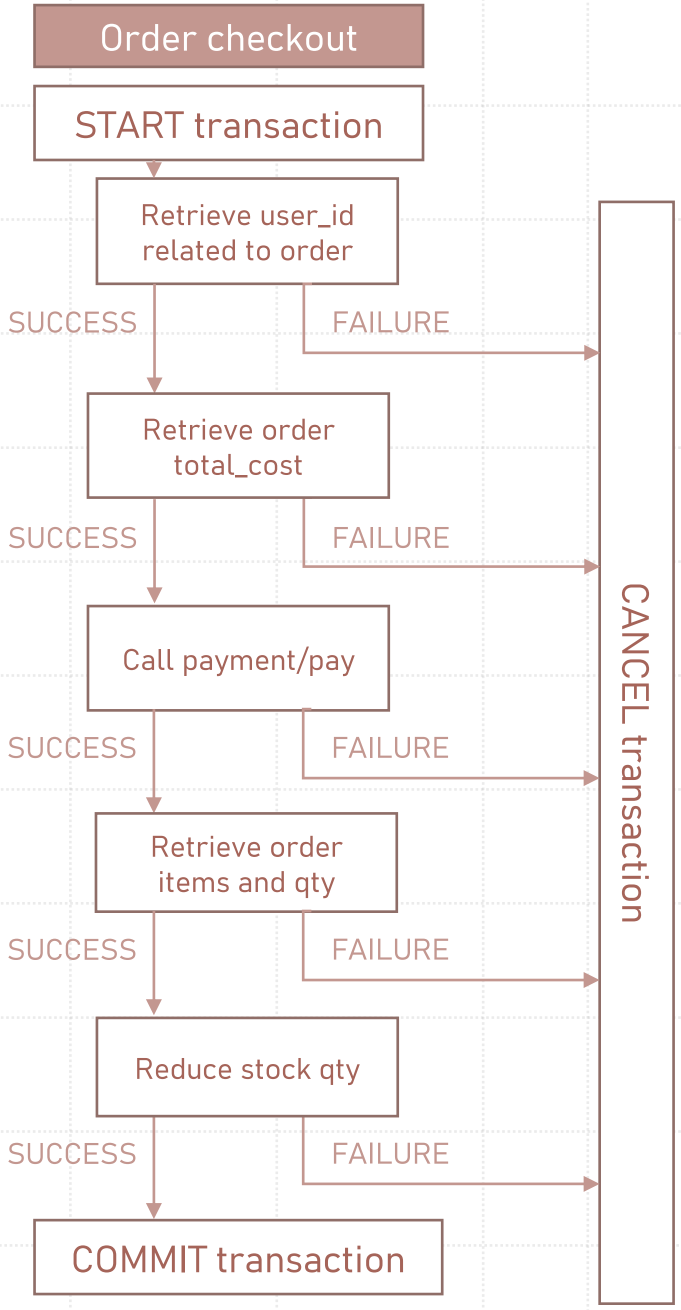
System architecture



Connection manager

- Coordinates requests that involve multiple micro-services
- Logic is used for requests that involve multiple queries as well, to ensure requests to endpoints are either fully processed or not processed at all





Transaction execution

- Using built-in Cockroach transaction support along with connection manager for coordination of multiple statement requests to achieve strong consistency
- CockroachDB guarantees ACID transactions for distributed transactions
- CockroachDB always uses serializable isolation

Consistency

- Cockroach DB is consistent across database replicas by using the Raft consensus algorithm for writes and a custom time-based synchronization algorithms for reads

Source: Cockroach Labs documentation

Fault tolerance

Business-code level

- Automatic restart of failed micro-services in Kubernetes

Data level

- Using built-in functionality of CockroachDB for failure recovery that ensures strong consistency across replicas

Scalability

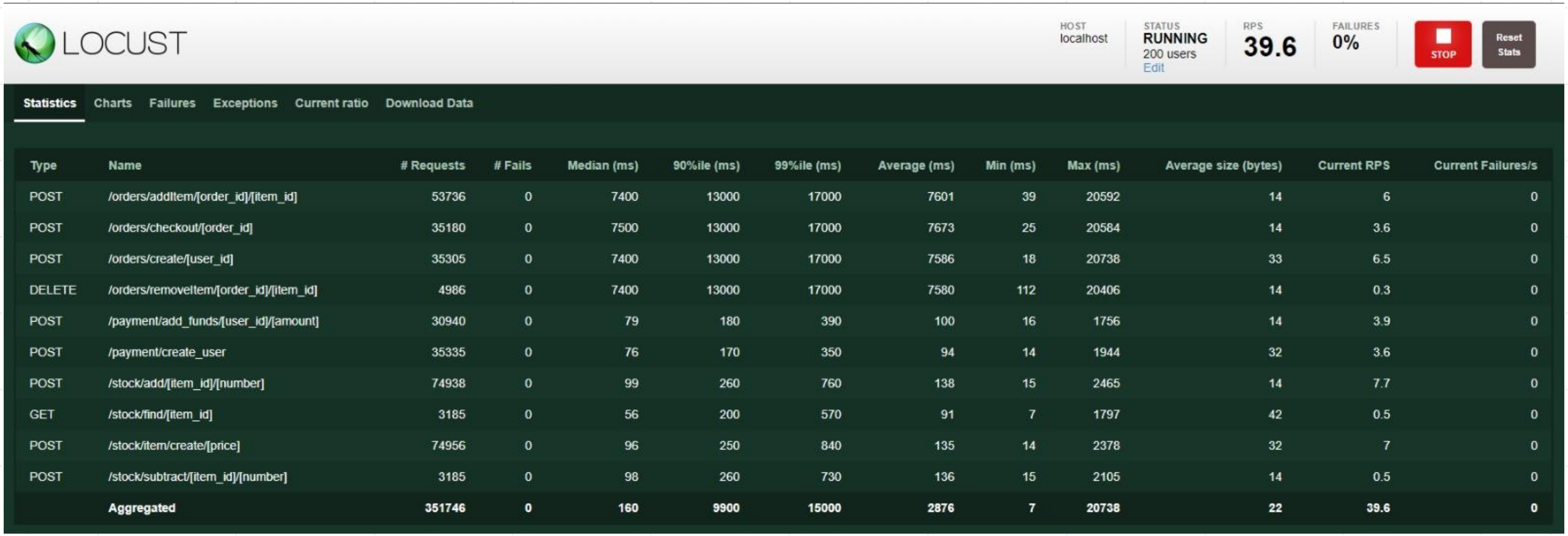
Business-code level

- Manual scaling to accommodate demand

Data level

- Manual scaling to accommodate demand
- Cockroach splits the key-value range when reaching the storage threshold and scales horizontally with automatic rebalancing and replication across nodes

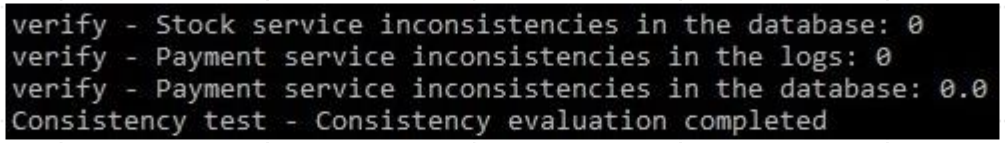
Results – Latency and throughput



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]	53736	0	7400	13000	17000	7601	39	20592	14	6	0
POST	/orders/checkout/[order_id]	35180	0	7500	13000	17000	7673	25	20584	14	3.6	0
POST	/orders/create/[user_id]	35305	0	7400	13000	17000	7586	18	20738	33	6.5	0
DELETE	/orders/removeItem/[order_id]/[item_id]	4986	0	7400	13000	17000	7580	112	20406	14	0.3	0
POST	/payment/add_funds/[user_id]/[amount]	30940	0	79	180	390	100	16	1756	14	3.9	0
POST	/payment/create_user	35335	0	76	170	350	94	14	1944	32	3.6	0
POST	/stock/add/[item_id]/[number]	74938	0	99	260	760	138	15	2465	14	7.7	0
GET	/stock/find/[item_id]	3185	0	56	200	570	91	7	1797	42	0.5	0
POST	/stock/item/create/[price]	74956	0	96	250	840	135	14	2378	32	7	0
POST	/stock/subtract/[item_id]/[number]	3185	0	98	260	730	136	15	2105	14	0.5	0
	Aggregated	**351746**	**0**	**160**	**9900**	**15000**	**2876**	**7**	**20738**	**22**	**39.6**	**0**

Results – Consistency

- No inconsistencies on consistency test at cost of high latency



```
verify - Stock service inconsistencies in the database: 0
verify - Payment service inconsistencies in the logs: 0
verify - Payment service inconsistencies in the database: 0.0
Consistency test - Consistency evaluation completed
```

Summary

Project strengths

- Strongly consistent design
- No need to locate data: queries can be sent to any replica of Cockroach database for processing
- Isolation of requests involving multiple microservices through connection manager

Project weaknesses

- Manual scaling instead of auto-scaling
- Strongly consistency at cost of higher latency
- No retry logic implemented to in case of request failure due to machine failure
- Only tested on local cluster