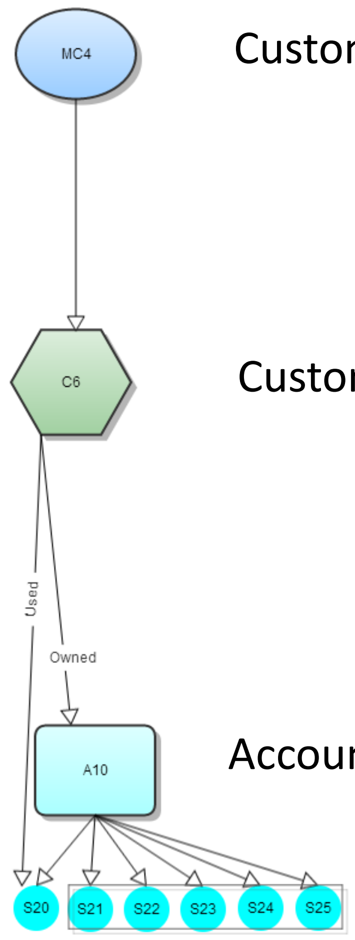
Using a graph database for resource authorization

Roar Audun Myrset and Sebastian Verheughe

2nd Workshop on Graph-based Technologies and Applications
«Graph-TA»
Barcelona, Spain
February 2014

Customer model



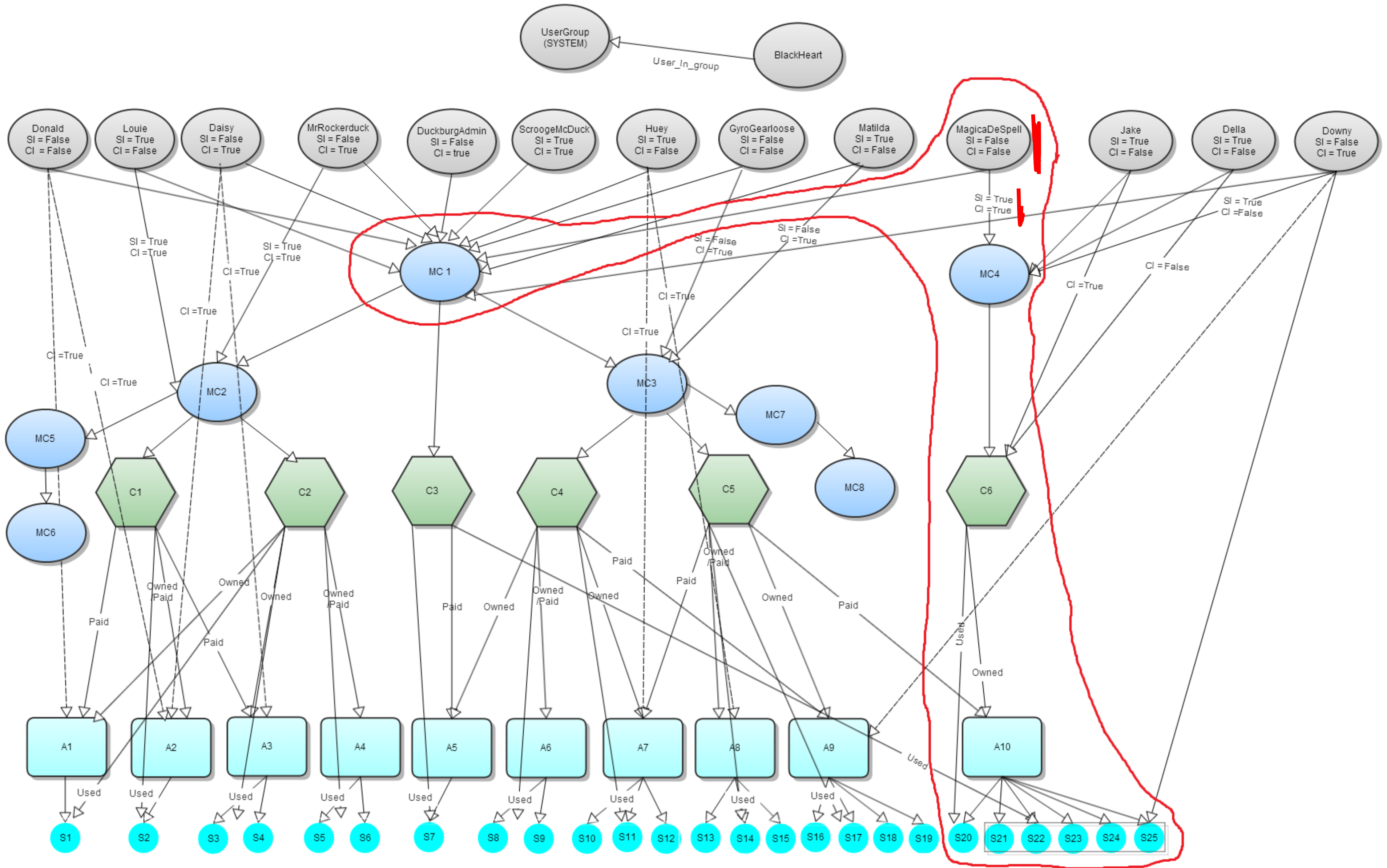
Customer

Customer

Account

Subscription(cell phone)

The problem : Which resources does a user have access to?


UserGroup (SYSTEM)
BlackHeart
User_In_group
Donald SI = False CI = False
Louie SI = True CI = False
Daisy SI = False CI = True
MrRockerduck SI = False CI = True
DuckburgAdmin SI = False CI = true
ScroogeMcDuck SI = True CI = True
Huey SI = True CI = False
GyroGearloose SI = False CI = False
Matilda SI = True CI = False
MagicaDeSpell SI = False CI = False
Jake SI = True CI = False
Della SI = True CI = False
Downy SI = False CI = True
SI = True CI =True
SI = False CI =True
SI = True CI =False
CI =True
CI =False
MC 1
MC2
MC3
MC4
MC5
MC6
MC7
MC8
C1
C2
C3
C4
C5
C6
Paid
Owned
Owned /Paid
Used
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
S22
S23
S24
S25

The problem domain

The resource authorization logic was written in SQL and had been running for several years.

But we started to get into trouble:

- Calculation of resources for a large user could take up to twenty minutes
- To get acceptable response times we needed to introduce caching which again caused:
  - Inability to scale for more users with access to a large set of resources
  - Data not being up to date
- Complicated to understand the SQL code

Solution part 1

- Question: Re-implement the existing SQL/relational database or make something new?
  - Decision was made to use a graph database based on gut feeling and a Proof of Concept
  - The graph database chosen was Neo4J

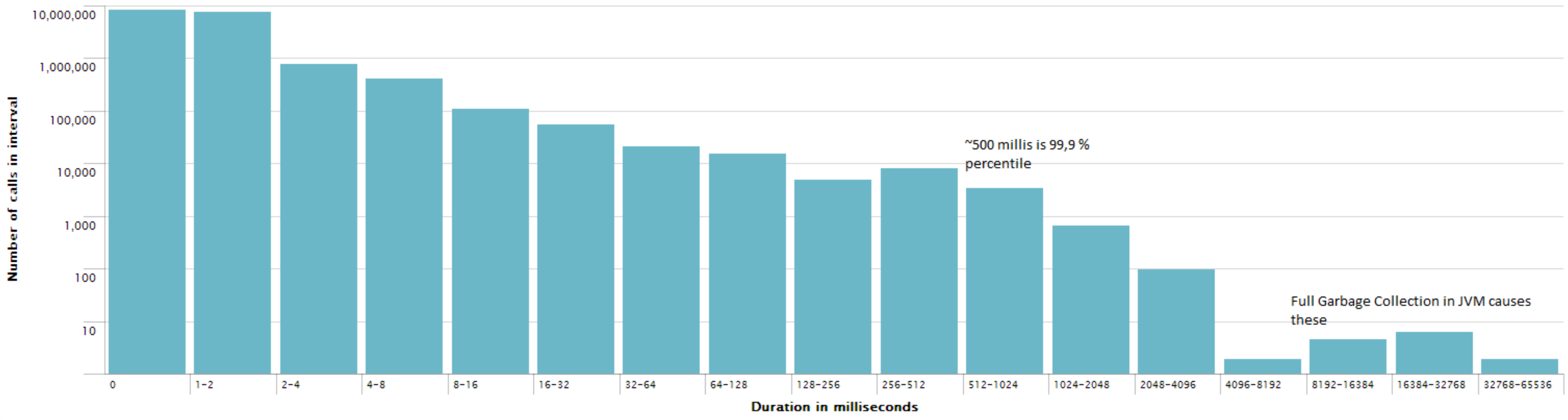
Solution part 2

- ~30 million nodes/vertices in the graph
  - ~3 million user nodes
  - ~27 million resource nodes
- ~40 million relationships/edges
- At runtime the database is loaded into memory (in-heap) and is using about 20GB.

Results

- We don’t need to cache anymore hence we are able to scale with regards to the number of users
- The graph query logic is easier to read than the old SQL logic.
- Users are getting acceptable response times:

Distribution of response times the last 30 days in production. Total of ~18 million calls



Reasons to meet us at the poster

- Experience with using Neo4J
- Performance optimizations done
- All other stuff we didn’t get to tell you