Multicom – FlightAware's Alert Delivery System

Mary Ryan Gilmore

TCL Conference 2019

What is Multicom?

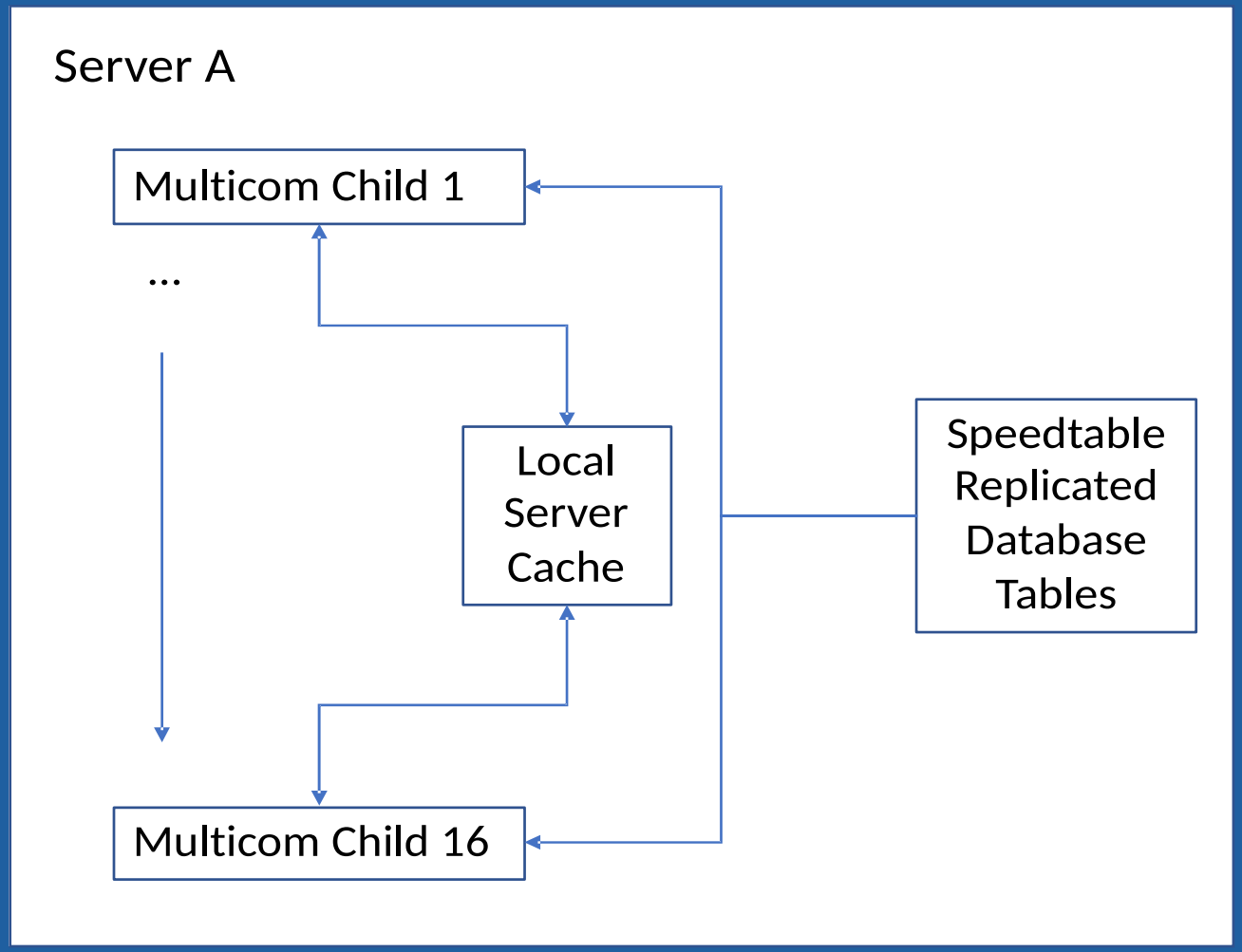
- High performance alert delivery application
- Reads in a stream of 35 million flight event messages per day (that's 400 updates per second on average)
- Matches the events against more than 440,000 alert triggers
- Sends more than 300,000 alerts each day

Using TCL Packages to Improve Performance

- As FlightAware gains access to more data sources, the number of events processed per day is only increasing
- Improved performance by introducing sqlite and sqlbird
- Improved durability by introducing tclrmq and zookeepertcl

Old Multicom Design

- Before sqlite and sqlbird were introduced, it ran on 3 servers with 16 children on each server (for 48 children total)



Problems with Speedtables Design

- Does not have an “OR” functionality like postgres
- There was a lot of write contention on the speedtable cache because all 16 of the children on one server where attempting to write to it

Response Following Sudden Event Data Increase

- We threw more servers at the problem
- We started running 96 child processes across 6 servers

Introducing Sqlite and Sqlbird

- Sqlite solved the “OR” problem
- (speedtables)

```
foreach field {base_id ident reg origin destination aircrafttype} {
    set search_list [list [list match $field $data($field)] {true enabled}]
    $::st(mc_trigger_tracking) search -compare $search_list -array trigger -code {
    …
    }
}
```

- (sqlbird)

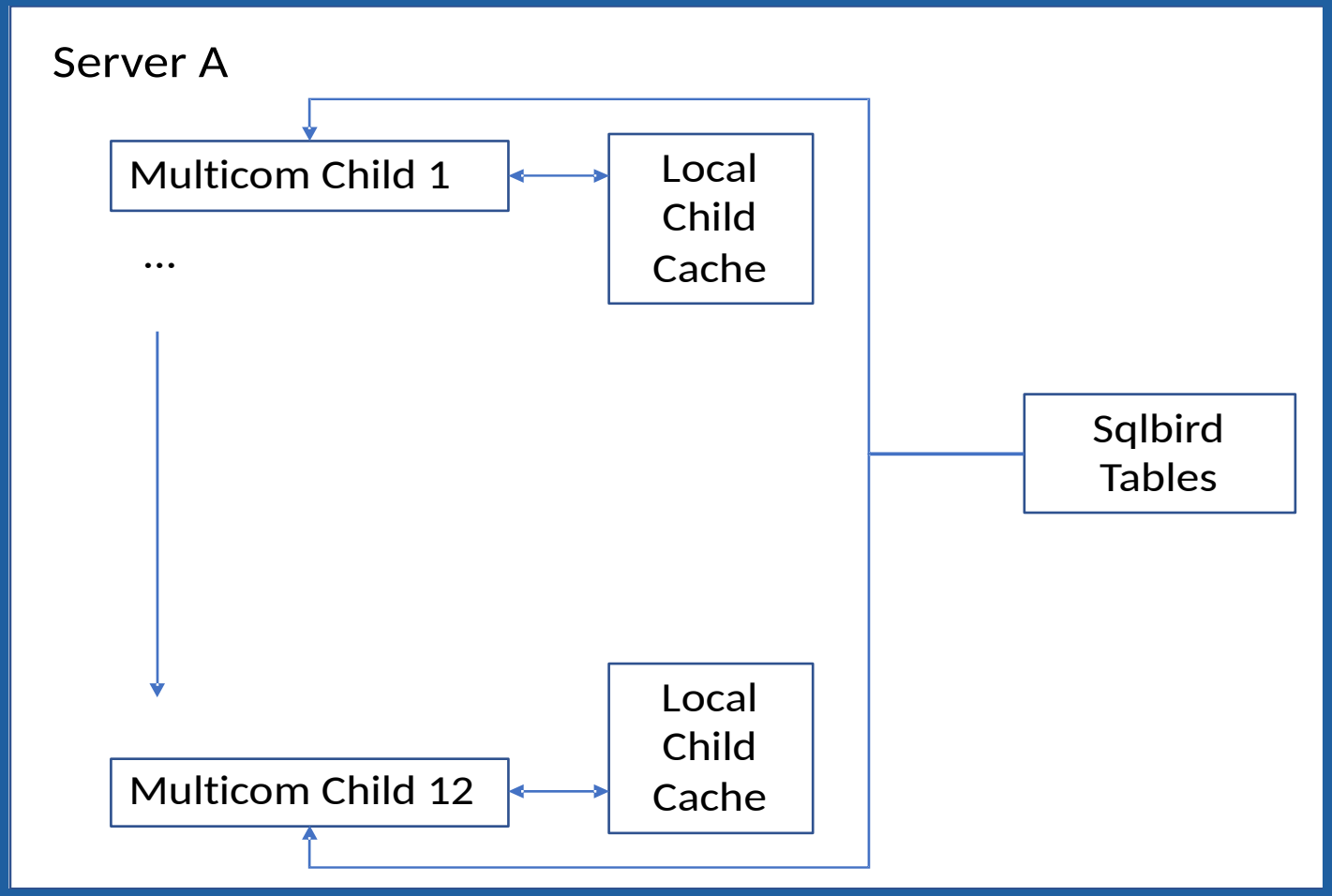
```
set sql “SELECT * FROM mc_trigger_tracking WHERE ident = :data(ident) OR reg = :data(reg) OR origin =
:data(origin) OR destination = :data(destination) OR aircrafttpye = :data(aircrafttype)”

sqlbird::select $sql trigger {

    ...
}
```

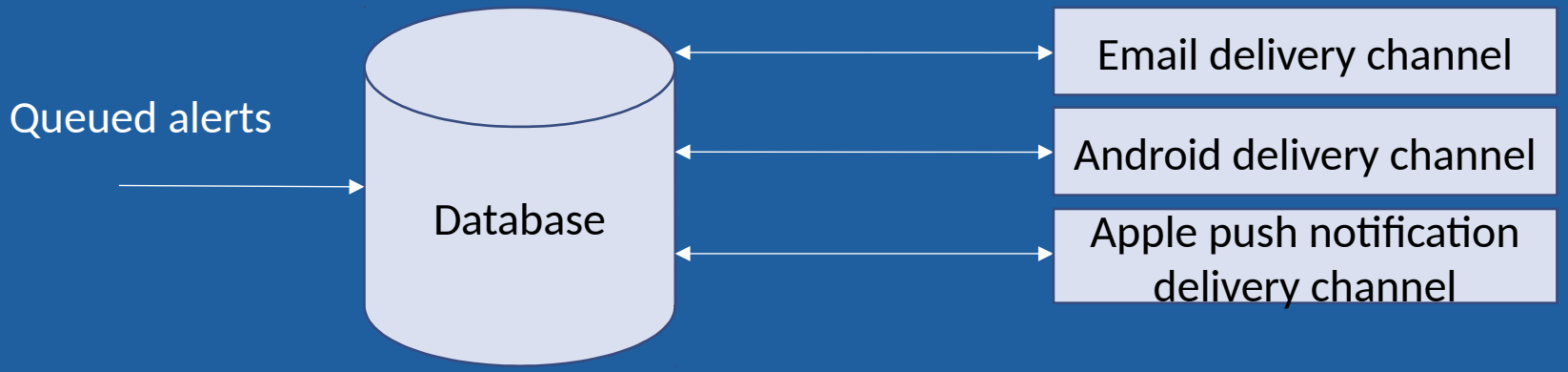
New Multicom Design

- After sqlite and sqlbird were introduced, it ran on 2 servers with 12 children on each server (for 24 children total)

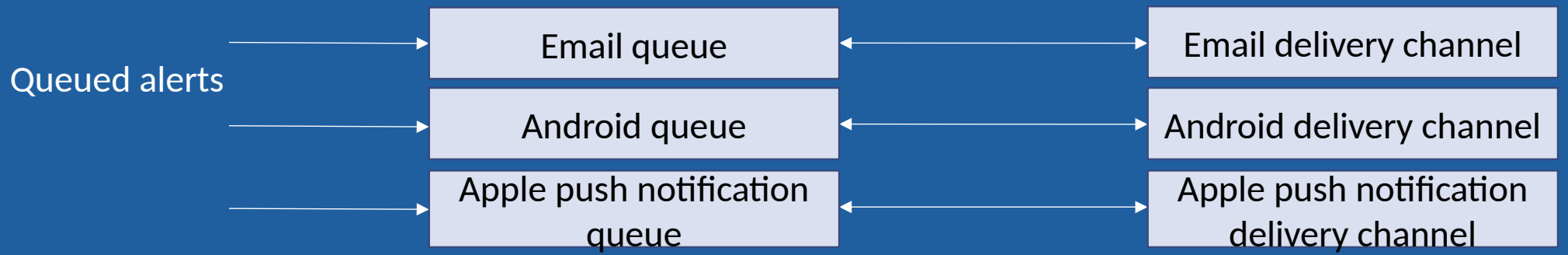


The New Problem: Database Dependence

- New company expectation that if the database goes down, your application does not
- Mutlicom was using a database table as a queue



Using Tclrmq to Enable us to Use RabbitMQ



Using Zookeepertcl to Store PITR

- Another thing that we used the database for was storing point in time references for each child process
- Moved this functionality to zookeeper

Conclusion

- We attribute a lot of Multicom’s continued growth and success to new TCL libraries such as sqlite, sqlbird, tclrmq, and zookeepertcl
- Look forward to learning about new ways that we can use TCL to improve this system