PAT HELLAND AND ME

HOW TO BUILD STATEFUL DISTRIBUTED APPLICATIONS THAT CAN SCALE ALMOST INFINITELY
SEAN T. ALLEN

VP OF ENGINEERING AT WALLAROO LABS
MEMBER OF THE PONY CORE TEAM
AUTHOR OF “STORM APPLIED”
 LOVER OF FRENCH STREET ART

@SEANTALLEN
@WALLAROOLABS
@PONYLANG
PAT HELLAND
AND ME
PAT HELLAND
WRITER OF PAPERS I LOVE
PAT HELLAND
LIFE BEYOND DISTRIBUTED TRANSACTIONS
WHAT’S IN THIS TALK...
WHAT IS SCALING?
MORE AND MORE THINGS

BUT, THEY DON’T GET BIGGER.
THERE’S JUST...

MORE OF THEM. LOTS MORE.
SOME AXIOMS...
TO SCALE INFINITELY, WE HAVE TO SCALE HORIZONTALLY
TO SCALE INFINITELY, WE MUST AVOID COORDINATION
DISTRIBUTED TRANSACTIONS ARE A FORM OF COORDINATION
THEREFORE...

TO SCALE INFINITELY, WE CAN’T USE TRANSACTIONS
WE SCALE ENTITIES

ENTITIES:

LIVE ON A SINGLE MACHINE
AND ARE MANIPULATED INDIVIDUALLY
WHAT IS AN ENTITY?
ENTITIES ARE BOUNDARIES OF ATOMICITY
DENORMALIZE...

ALL THE THINGS!
TWO-LAYER ARCHITECTURE
scale-agnostic

API

scale-aware
scale-agnostic

API

scale-aware
TO SCALE INFINITELY, YOUR BUSINESS LOGIC HAS TO BE INDEPENDENT OF SCALE
TWO BIG IDEAS
A WORLD OF POSSIBILITIES
ENTITIES

BUT WE CALL THEM…

“STATE OBJECTS”
TWO-LAYER ARCHITECTURE

BUT WE CALL IT...

“SCALE INDEPENDENCE”
user supplied logic

Wallaroo API

Wallaroo runtime
user supplied logic

Wallaroo API

Wallaroo runtime
Wallaroo runtime

Wallaroo API

user supplied logic
WALLAROO API
MARKET SPREAD EXAMPLE
MARKET SPREAD
REAL-TIME “SOMETHING AIN’T RIGHT” TRADE CHECKS
MARKET SPREAD
TWO SOURCES OF DATA
MARKET SPREAD

MARKET DATA PIPELINE

Market Data -> Update APPL

Orders -> Check MSFT

Rejections

Market Spread

State
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder)

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
```python
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
```
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, symbol_data, symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder)

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, symbol_data, symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
               wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
                      symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder)

ab.new_pipeline("Market Data",
               wallaroo.TCPSourceConfig(nhbo_host, nhbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
                      symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
)

ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbho_host, nbho_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
Define our sinks

Orders pipeline might have output

```
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
```
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
ab = wallaroo.ApplicationBuilder("Market Spread")

ab.new_pipeline("Orders",
    wallaroo.TCPSourceConfig(order_host, order_port, order_decoder)
ab.to_state_partition(check_order, SymbolData, "symbol-data",
    symbol_partition_function)
ab.to_sink(wallaroo.TCPSinkConfig(out_host, out_port, order_result_encoder))

ab.new_pipeline("Market Data",
    wallaroo.TCPSourceConfig(nbbo_host, nbbo_port, market_data_decoder)
ab.to_state_partition(update_market_data, SymbolData, "symbol-data",
    symbol_partition_function)
ab.done()
USER SUPPLIED
LOGIC
@wallaroo.state_computation(name="Update Market Data")
def update_market_data(data, state):
    offer_bid_difference = data.offer - data.bid

    should_reject_trades = ((offer_bid_difference >= 0.05) or
                           ((offer_bid_difference / data.mid) >= 0.05))

    state.last_bid = data.bid
    state.last_offer = data.offer
    state.should_reject_trades = should_reject_trades

    return (None, True)
WALLAROO RUNTIME

MESH NETWORK OF COOPERATING PROCESSES
STATE OBJECTS

ONE BIG MAP?
STATE OBJECTS
CONCEPTUALLY IT'S LIKE A BIG MAP
STATE OBJECTS
WITH A KEY FOR EACH OBJECT

Market Data

APPL
IBM
MSFT
AMZN
INTC
NVDA

Update
STATE OBJECTS
WHERE WE MAY FROM INCOMING DATA'S KEY
STATE OBJECTS

TO THE STATE OBJECT FOR THAT KEY
HASH PARTITIONING
DISTRIBUTING STATE OBJECTS ACROSS A CLUSTER
SINGLE WORKER

ALL SYMBOLS TOGETHER

APPL
IBM
MSFT
AMZN
SINGLE WORKER
ALL SYMBOLS TOGETHER
ADD ANOTHER WORKER

STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER

STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER

STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER
STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER

STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER
STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
ADD ANOTHER WORKER

STATE OBJECTS WILL BE REDISTRIBUTED ACROSS THE CLUSTER
STATE OBJECTS
class SymbolData(object):
    def __init__(self, last_bid=0.0, last_offer=0.0, should_reject_trades=True):
        self.last_bid = last_bid
        self.last_offer = last_offer
        self.should_reject_trades = should_reject_trades
LEARN MORE

GITHUB.COM/SEANTALLEN/PAT-HELLAND-AND-ME