VAST: Interactive Network Forensics

Matthias Vallentin
matthias@bro.org

BroCon
August 5, 2015
Demo I
Data Pyramid

- Alarms
- Bro Logs
- Bro Events
- Packets

Data Volume

High Fidelity

Low Fidelity
Data Pyramid

- Exit Status
- Process Events
- System Calls
- Instruction Stream

Low Fidelity

High Fidelity

Data Volume

Data Volume

High Fidelity

Low Fidelity
VAST: Visibility Across Space and Time

Key Features

- Interactive response times
- Horizontal scaling over a cluster
- Iterative query refinement
- Type-rich data model
- Strongly typed query language
- Historical & continuous queries
High-Level Architecture of VAST

Import

- Sources produce events
- PCAP, Bro logs, BGPdump, ...
High-Level Architecture of VAST

Import
- Sources produce events
- PCAP, Bro logs, BGPdump, ...

Archive
- Key-value store (IDs → events)
- Stores raw data as events
High-Level Architecture of VAST

**Import**
- Sources produce events
- PCAP, Bro logs, BGPdump, ...

**Archive**
- Key-value store (IDs → events)
- Stores raw data as events

**Index**
- Bitmap indexes over event data
- Hits are event IDs in archive
High-Level Architecture of VAST

Import
- Sources produce events
- PCAP, Bro logs, BGPdump, ...

Archive
- Key-value store (IDs → events)
- Stores raw data as events

Index
- Bitmap indexes over event data
- Hits are event IDs in archive

Export
- Sinks consume events
- PCAP, Bro logs, ASCII, JSON
# VAST & Big Data

## MapReduce (Hadoop)

<table>
<thead>
<tr>
<th>Batch-oriented processing: full scan of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Expressive: no restriction on algorithms</td>
</tr>
<tr>
<td>- Speed &amp; Interactivity: full scan for each query</td>
</tr>
</tbody>
</table>
# VAST & Big Data

## MapReduce (Hadoop)

- Batch-oriented processing: *full scan* of data
  - Expressive: no restriction on algorithms
  - Speed & Interactivity: full scan for each query

## In-memory Cluster Computing (Spark)

- Load full data set into memory and then run query
  - Speed & Interactivity: fast on arbitrary queries over working set
  - Thrashing when working set too large
# VAST & Big Data

## MapReduce (Hadoop)

Batch-oriented processing: *full scan* of data

- Expressive: no restriction on algorithms
- Speed & Interactivity: full scan for each query

## In-memory Cluster Computing (Spark)

Load full data set into memory and then run query

- Speed & Interactivity: fast on arbitrary queries over working set
- Thrashing when working set too large

## Distributed Indexing (VAST)

Distributed building and querying of bitmap indexes

- Fast: only access space-efficient indexes
- Caching of index hits enables iterative analyses
- Lookup only, not arbitrary computation
<table>
<thead>
<tr>
<th>Splunk</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>Unstructured text</td>
</tr>
<tr>
<td>Index</td>
<td>B-tree</td>
</tr>
<tr>
<td>Computation</td>
<td>MapReduce</td>
</tr>
<tr>
<td>Code</td>
<td>Closed-source</td>
</tr>
<tr>
<td>License</td>
<td>Data-volume based</td>
</tr>
<tr>
<td>Splunk</td>
<td>ElasticSearch</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Data Model</td>
<td>Data Model</td>
</tr>
<tr>
<td>Unstructured text</td>
<td>Rich (Lucene)</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>B-tree</td>
<td>Inverted (Lucene)</td>
</tr>
<tr>
<td>Computation</td>
<td>Computation</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Index Lookup</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
</tr>
<tr>
<td>Closed-source</td>
<td>Open-source</td>
</tr>
<tr>
<td>License</td>
<td>License</td>
</tr>
<tr>
<td>Data-volume based</td>
<td>Apache 2.2</td>
</tr>
<tr>
<td>Splunk</td>
<td>ElasticSearch</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Data Model</td>
<td>Data Model</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Computation</td>
<td>Computation</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
</tr>
<tr>
<td>License</td>
<td>License</td>
</tr>
<tr>
<td>Unstructured text</td>
<td>Rich (Lucene)</td>
</tr>
<tr>
<td>B-tree</td>
<td>Inverted (Lucene)</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Open-source</td>
</tr>
<tr>
<td>Closed-source</td>
<td>Index Lookup</td>
</tr>
<tr>
<td>Data-volume based</td>
<td>Apache 2.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Computation</td>
</tr>
<tr>
<td>Code</td>
</tr>
<tr>
<td>License</td>
</tr>
</tbody>
</table>
Types: Interpretation of Data

- **Types**:
  - Interpretation of Data

- **Basic Types**:
  - bool
  - int
  - count
  - real
  - duration
  - time
  - string
  - pattern
  - address
  - subnet
  - port
  - none

- **Compound Types**:
  - vector
  - set
  - table

- **Recursive Types**:
  - record

- **Container Types**:
  - field 1
  - ... field n

```
record
  field 1
  ... field n
```
# Query Language

## Boolean Expressions
- **Conjunctions** `&&`
- **Disjunctions** `||`
- **Negations** `!`
- **Predicates**
  - `LHS op RHS`
  - `(expr)`

## Examples
- `A && B || !(C && D)`
- `orig_h == 10.0.0.1 && &time < now - 2h`
- `&type == "conn" || "foo" in :string`
- `duration > 60s && service == "tcp"`

## Extractors
- `&type`
- `&time`
- `x.y.z.arg`
- `:type`

## Relational Operators
- `<, <=, ==, >=, >`
- `in, ni, [+ , +]`
- `!in, !ni, [-, -]`
- `~, ~`

## Values
- `T, F`
- `+42, 1337, 3.14`
- "foo"
- `10.0.0.0/8`
- `80/tcp, 53/?`
- `{1, 2, 3}`
Index Hits: Sets of Event IDs

**Bitvector:** ordered set of IDs
- Query result $\equiv$ set of event IDs from $[0, 2^{64} - 1)$
- Model as **bit vector**: $[4, 7, 8] = 0000100110\ldots$
- Run-length encoded
- Append-only
- Bitwise operations do not require decoding

**Bitmap:** maps values to bit vectors
- `push_back(T x)`: append value $x$ of type $T$
- `lookup(T x, Op ◦)`: get bit vector for $x$ under $\circ$
Composing Results via Bitwise Operations

Combining Predicates

- Query $Q = X \land Y \land Z$
  - $x = 1.2.3.4 \land y < 42 \land z \in \text{"foo"}$
- Bitmap index lookup yields $X \to B_1$, $Y \to B_2$, and $Z \to B_3$
- Result $R = B_1 \& B_2 \& B_3$
What happened since BroCon’14?

**New Features**

- Continuous queries
  - Apply queries to arriving data
- Time Machine
- Full indexes on time stamp and connection tuple
- Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
- Distributed Architecture
  - Commutativity: support message reordering
  - Associativity: parallel query engine
What happened since BroCon’14?

**New Features**

- Continuous queries
  - Apply queries to arriving data
- Time Machine
  - Full indexes on time stamp and connection tuple
  - Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
- Distributed Architecture
  - Commutativity: support message reordering
  - Associativity: parallel query engine
What happened since BroCon’14?

New Features

- Continuous queries
  - Apply queries to arriving data
- Time Machine
  - Full indexes on time stamp and connection tuple
  - Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
What happened since BroCon’14?

New Features

- Continuous queries
  - Apply queries to arriving data
- Time Machine
  - Full indexes on time stamp and connection tuple
  - Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
- Distributed Architecture
  - Commutativity: support message reordering
  - Associativity: parallel query engine
What happened since BroCon’14?

New Features

- Continuous queries
  - Apply queries to arriving data
- Time Machine
  - Full indexes on time stamp and connection tuple
  - Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
- Distributed Architecture
  - Commutativity: support message reordering
  - Associativity: parallel query engine
**Distributed VAST**

**NODE**: the logical unit of deployment

- A container for actors/components
- Message serialization only at NODE boundaries
- Maps to single OS process, typically one per machine
Distributed VAST: Replicated Cores
Distributed VAST: Replicated Cores

SOURCE

SINK

A
E
I
X
Distributed VAST: Custom Deployment
Demo II
Demo Topology: Import
Demo Topology: Import

SOURCE

FOO

BAR

I
A
X

I
A
X

ID

foo
bar

ID
source
Demo Topology: Export (naive)
Demo Topology: Export (better)
Demo Topology: Export (good)
Future Work: Moving Forward

**Next Milestone: Release**

- Architecture converging: feature freeze for 0.1 soon
- Thorough testing of distributed architecture
- Improve index size of strings and containers

**Down The Line**

- Improved Bro integration
- Unify data model with Broker
- VAST writer for Bro
- Fault tolerance
- Data replication (replicate archive & index)
- Query snapshotting (resume failed execution)
- Use Raft to manage global state (large-scale clusters)
- Interface with Spark to enable arbitrary computation
- Interface with Spicy for powerful event import/export
## Future Work: Moving Forward

### Next Milestone: Release
- Architecture converging: feature freeze for 0.1 soon
- Thorough testing of distributed architecture
- Improve index size of strings and containers

### Down The Line
- Improved Bro integration
  - Unify data model with Broker
  - VAST writer for Bro
- Fault tolerance
- Data replication (replicate archive & index)
- Query snapshotting (resume failed execution)
- Use Raft to manage global state (large-scale clusters)
- Interface with Spark to enable arbitrary computation
- Interface with Spicy for powerful event import/export
## Future Work: Moving Forward

### Next Milestone: Release
- Architecture converging: feature freeze for 0.1 soon
- Thorough testing of distributed architecture
- Improve index size of strings and containers

### Down The Line
- Improved Bro integration
  - Unify data model with Broker
  - VAST writer for Bro
- Fault tolerance
  - Data replication (replicate `ARCHIVE & INDEX`)
  - Query snapshotting (resume failed execution)
  - Use Raft to manage global state (large-scale clusters)
Future Work: Moving Forward

Next Milestone: Release
- Architecture converging: feature freeze for 0.1 soon
- Thorough testing of distributed architecture
- Improve index size of strings and containers

Down The Line
- Improved Bro integration
  - Unify data model with Broker
  - VAST writer for Bro
- Fault tolerance
  - Data replication (replicate archive & index)
  - Query snapshotting (resume failed execution)
  - Use Raft to manage global state (large-scale clusters)
- Interface with Spark to enable arbitrary computation
## Future Work: Moving Forward

### Next Milestone: Release
- Architecture converging: feature freeze for 0.1 soon
- Thorough testing of distributed architecture
- Improve index size of strings and containers

### Down The Line
- Improved Bro integration
  - Unify data model with Broker
  - VAST writer for Bro
- Fault tolerance
  - Data replication (replicate `ARCHIVE & INDEX`)
  - Query snapshotting (resume failed execution)
  - Use Raft to manage global state (large-scale clusters)
- Interface with Spark to enable arbitrary computation
- Interface with Spicy for powerful event import/export
Questions?

More at:
http://vast.tools