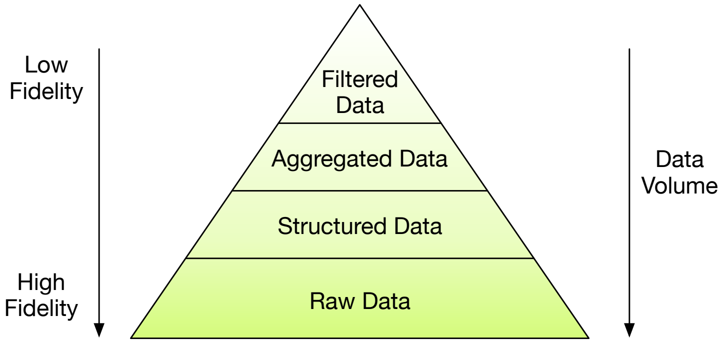
VAST: Interactive Network Forensics

Matthias Vallentin
matthias@bro.org

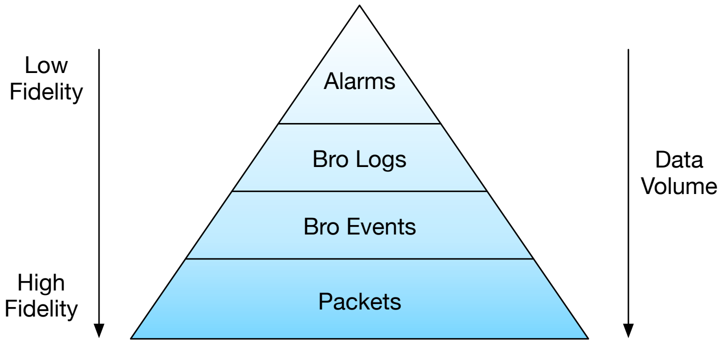
BroCon
August 5, 2015

Demo I

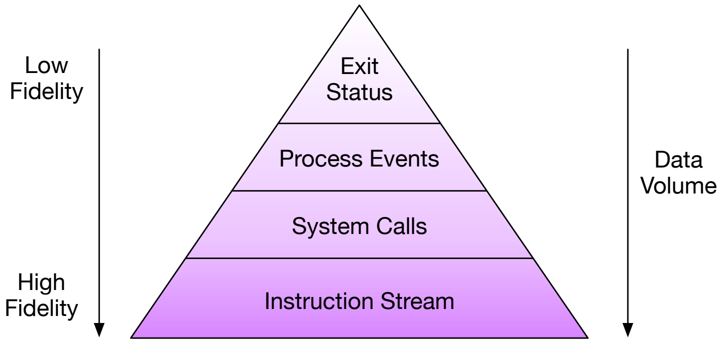
Data Pyramid



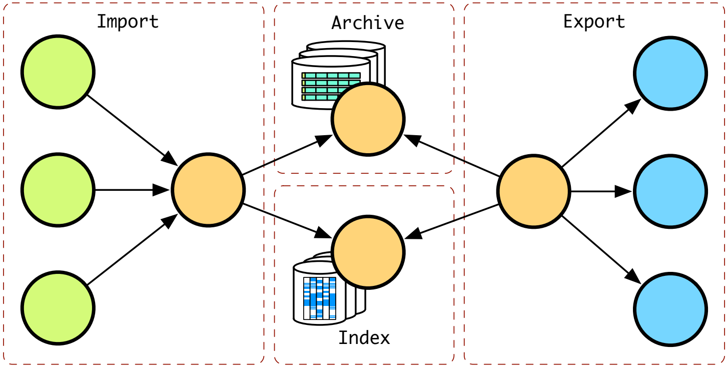
Data Pyramid



Data Pyramid



VAST: Visibility Across Space and Time



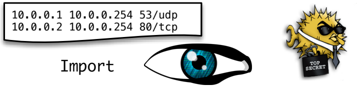
Key Features

- 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, . . .



```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

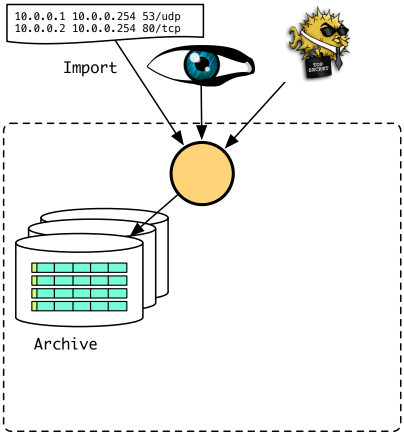
High-Level Architecture of VAST

Import

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

Archive

- Key-value store (IDs $\rightarrow$ events)
- Stores raw data as events



High-Level Architecture of VAST

Import

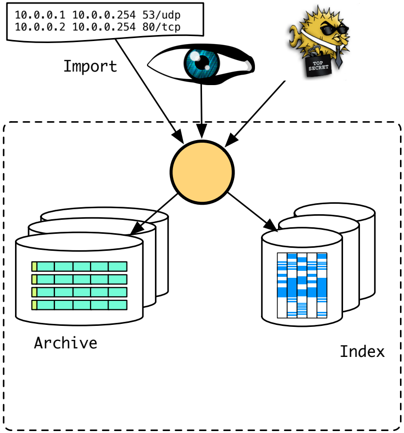
- Sources produce events
- PCAP, Bro logs, BGPdump, . . .

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

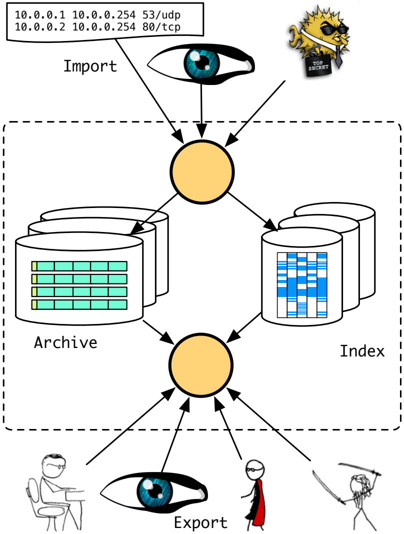
- Key-value store (IDs → events)
- Stores raw data as events

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)

Batch-oriented processing: *full scan* of data

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

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

VAST & SIEM

Splunk

Data Model	Unstructured text
Index	B-tree
Computation	MapReduce
Code	Closed-source
License	Data-volume based

VAST & SIEM

Splunk

Data Model	Unstructured text
Index	B-tree
Computation	MapReduce
Code	Closed-source
License	Data-volume based

ElasticSearch

Data Model	Rich (Lucene)
Index	Inverted (Lucene)
Computation	Index Lookup
Code	Open-source
License	Apache 2.2

VAST & SIEM

Splunk

Data Model	Unstructured text
Index	B-tree
Computation	MapReduce
Code	Closed-source
License	Data-volume based

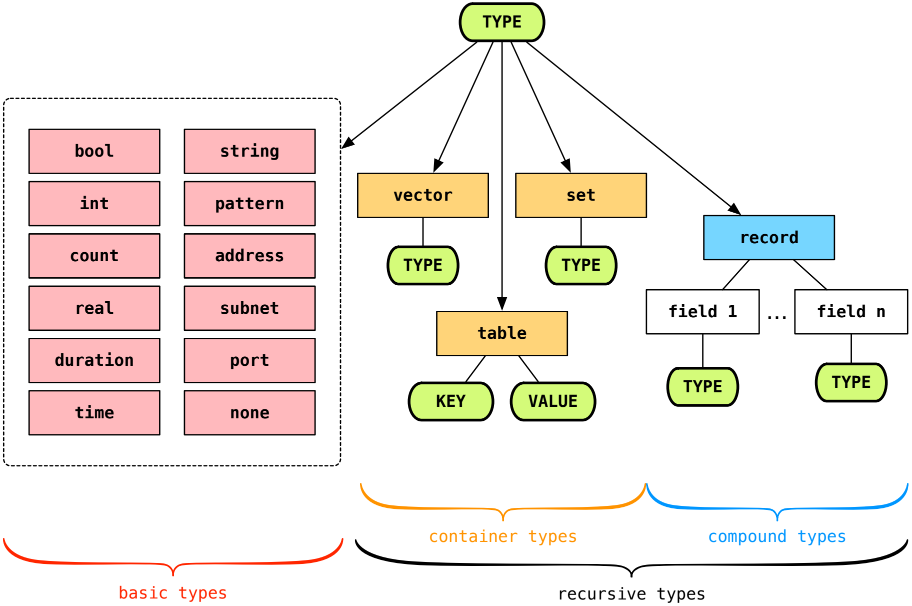
ElasticSearch

Data Model	Rich (Lucene)
Index	Inverted (Lucene)
Computation	Index Lookup
Code	Open-source
License	Apache 2.2

VAST

Data Model	Rich (Bro)
Index	Bitmap Indexes
Computation	Index Lookup
Code	Open-source
License	BSD (3-clause)

Types: Interpretation of Data



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}`

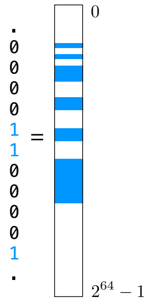
Index Hits: Sets of Event IDs

Bitvector: ordered set of IDs

- Query result $\equiv$ set of event IDs from $[0, 2^{64} - 1)$

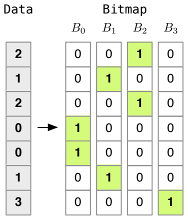
$\rightarrow$ Model as **bit vector**: $[4, 7, 8] = 0000100110\cdots$

- 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` $\circ$`)`: get bit vector for x under $\circ$

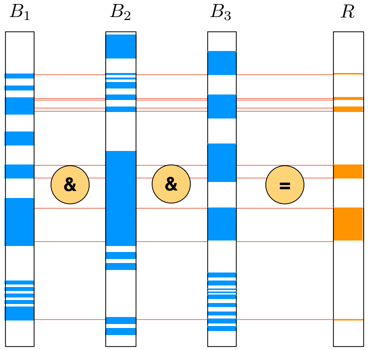


Data	B_0	B_1	B_2	B_3
2	0	0	1	0
1	0	1	0	0
2	0	0	1	0
0	1	0	0	0
0	1	0	0	0
1	0	1	0	0
3	0	0	0	1

Composing Results via Bitwise Operations

Combining Predicates

- Query $Q = X \wedge Y \wedge Z$
  - $x = \texttt{1.2.3.4} \ \wedge \ y < \texttt{42} \ \wedge \ z \in \texttt{"foo"}$
- Bitmap index lookup yields $X \rightarrow B_1$, $Y \rightarrow B_2$, and $Z \rightarrow B_3$
- Result $R = B_1 \ \& \ B_2 \ \& \ B_3$



What happened since BroCon’14?

New Features

- Continuous queries
  - Apply queries to arriving data

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

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

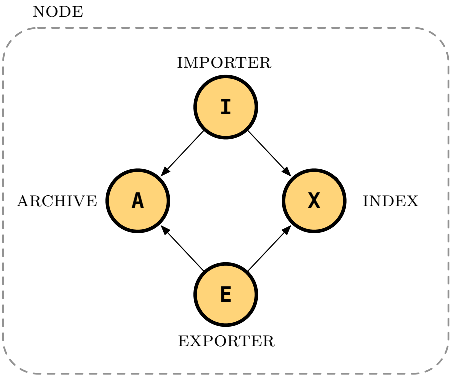
- Continuous queries
  - Apply queries to arriving data
- Time Machine
  - Full indexes on time stamp and connection tuple
  - Bidirectional flow cut-off
- New event sources
  - BGPdump
  - JSON/Kafka (not yet merged)
- Distributed Architecture
  - Commutativity: support message reordering
  - Associativity: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

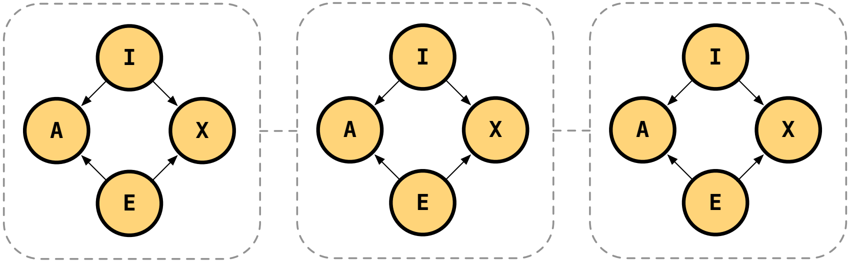


NODE: the logical unit of deployment

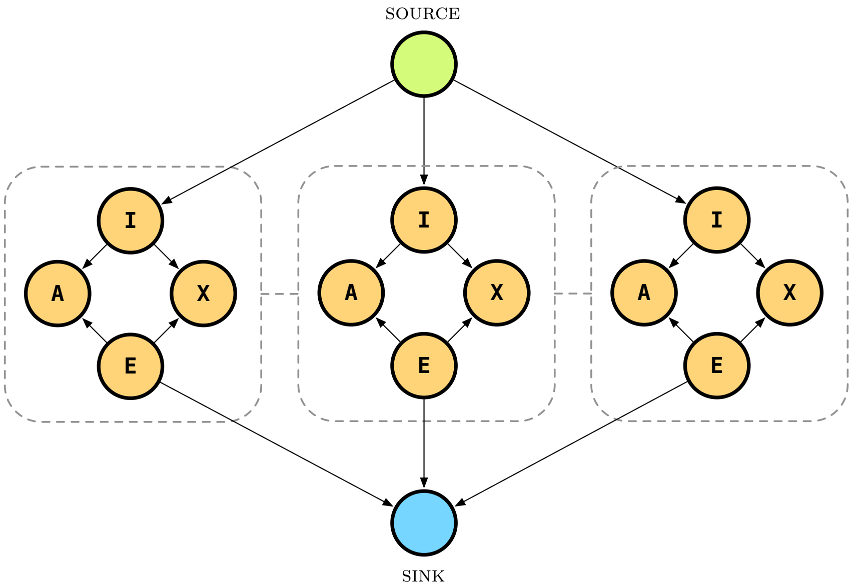
- A container for actors/components
- Message serialization only at NODE boundaries

→ Maps to single OS process, typically one per machine

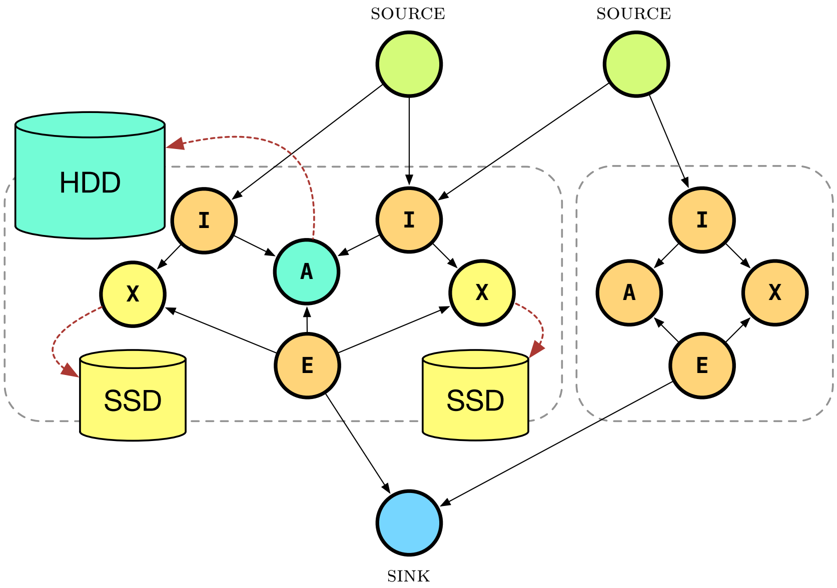
Distributed VAST: Replicated Cores



Distributed VAST: Replicated Cores

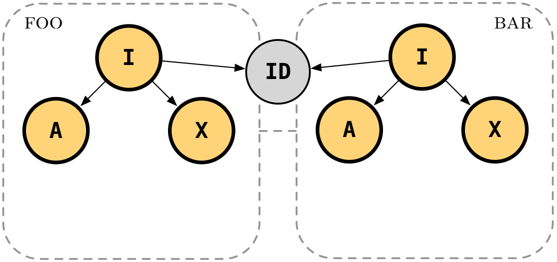


Distributed VAST: Custom Deployment

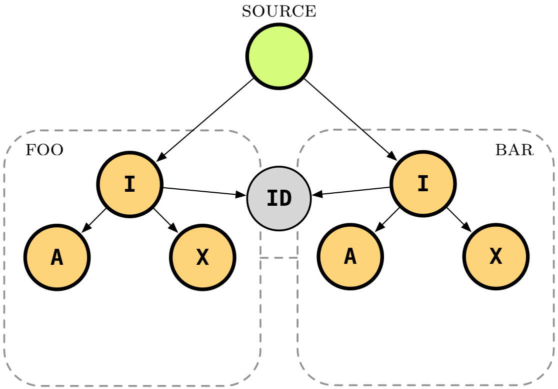


Demo II

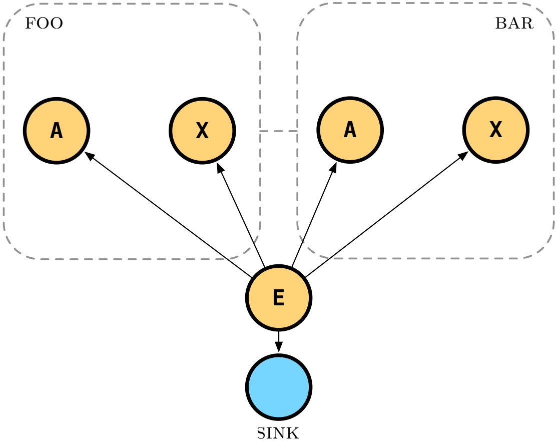
Demo Topology: Import



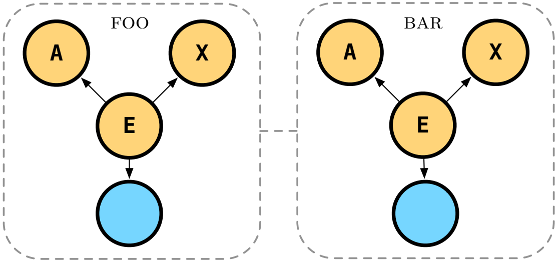
Demo Topology: Import



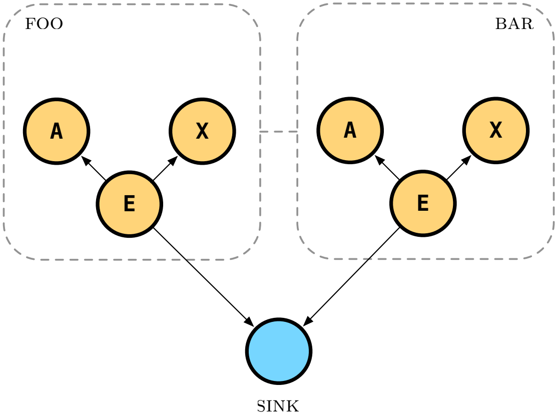
Demo Topology: Export (naive)



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

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

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