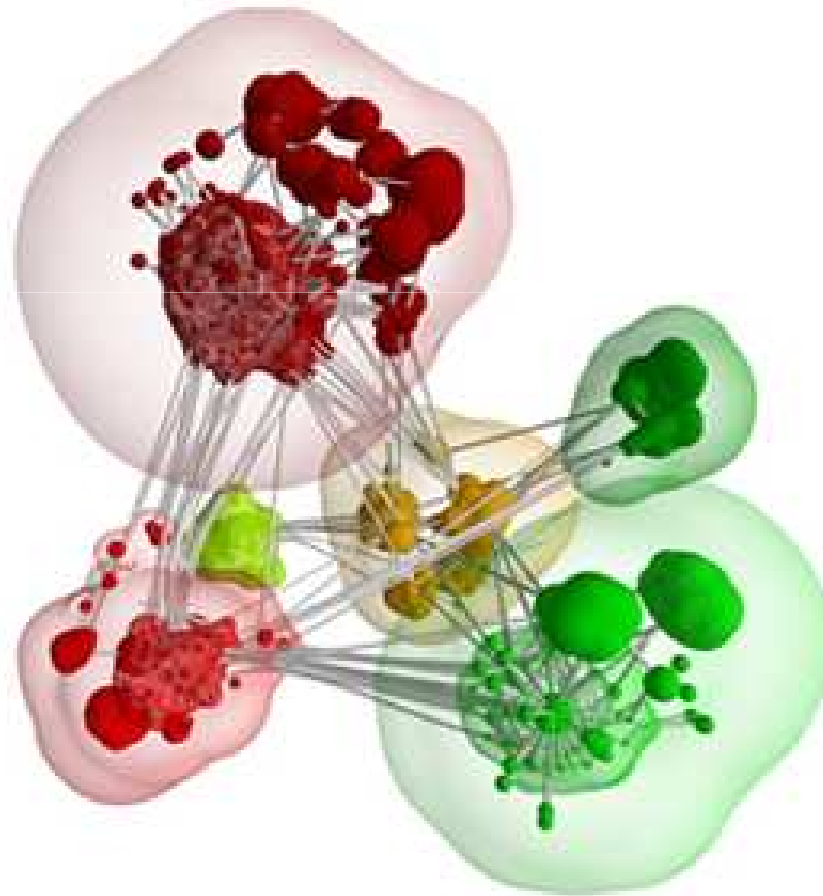




A triple store learns how to get into the Enterprise



Jans Aasman, Ph.D.
CEO Franz Inc
Ja@Franz.com



Franz Inc.

- Private, founded 1984 - U.C. Berkeley
- Self-funded and profitable
- Software tools for Artificial Intelligence (last 25)
- Relational Database Support (last 15 years)
- Object Oriented Database (last 10 years)
- AllegroGraph (RDF) Database (last 5 years)
- Major public and industry clients/industry



Thanksgiving 2005

- Version 0.1 (a prototype for a DOD conference)
- Written in Tokyo Dome Hotel
- 1103 lines (well, + 8010 lines for AllegroCache Btrees)
- Loaded 10,000 triples per second (industry record)
- Just Prolog and Lisp as a query language
- No server/client, no transactions
- No Sparql
- Died at roughly 200 M triples



2006

- Version 1.0
- Some DOD customers bought a license

- First SPARQL and SPARQL endpoint
- First version of prolog Select
- First time we used sort/merge approach
- Loaded at 14,000 triples per second

- No Transactions
- No range queries! ←



2007

- Version 2.0
- Same DOD customers bought license again
- Range queries: using native datatypes (numbers, dates, lat/longs, etc)
- RDFS++ reasoning
- Client/Server approach
 - Java clients for Sesame and Jena
- Integration with Top Braid Composer
- A little bit of Federation
- No Transactions, no read concurrency, no automatic indexing.

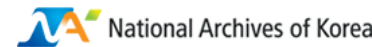


2008

- Version 3.0 to 3.3: A professional version of 2.0
- Same DOD customers bought license again 😊
- Query optimizer for Select (prolog)
 - Temporarily fastest reasoner without materialization
- GeoSpatial/Temporal reasoning, Social Network Analysis
- Freetext indexing
- Federation
- New http REST protocol -> Java, Python, Ruby, C#, Perl
- Gruff as graphical user interface
- No Transactions, no read concurrency, no automatic indexing.



The Redlands Institute





What our enterprise customers screamed for

- Transactions
 - ACID
 - Commit, Rollback
 - Check Pointing, Full and fast Recoverability.
- Transparent/Automatic Indexing
- Read and Write concurrency
- Lucene style indexing per predicate
- Clustering (use all cpus, memory, disks)
- Duplicate triple removal
- Garbage collection of deleted triples
- One click install
- Incremental Backup
- Warm Failover



What we worked on for the last two years and what we get with 4.0 (next week)

- ✓ Transactions
 - ✓ ACID
 - ✓ Commit, Rollback
 - ✓ Check Pointing, Full and fast Recoverability.
- ✓ Transparent/Automatic Indexing
- ✓ Read and Write concurrency
- ✓ Lucene style indexing per predicate
- ✓ Clustering (use all cpus, memory, disks)
- ✓ Duplicate triple removal
- ✓ Garbage collection of deleted triples
- ✓ One click install (well, for the server that works, and edit config file)
- Incremental Backup (summer)
- Replication and Warm Failover (summer)



Performance example: LUBM 8000

- Machine: 16 Gig machine, 2 disks, with 4 cores (2000)
- Indices: spogi and posgi
- One step load in 2 hours,
 - No preprocessing of strings, no indexing phase, no materialization
 - Ready to go query and reason
- If you want faster speed for LUBM queries then 45 minutes more indexing



However more important

- Customers want fair performance for parallel
 - Adding triples
 - Deleting triples
 - Queries



So we encouraging adoption of an Events Test Benchmark

- Goal: how good is the database in adding, deleting and queries triples at the same time and after the first X million triples have been added

- A proposed first version of the events test is at

<http://github.com/franzinc/agraph-python/blob/master/stress/events/events>



Search GitHub...

Home Pricing and Signup Explore GitHub Blog Login

franzinc / agraph-python

Watch Fork Download Source 3 2

Source Commits Network (2) Issues (0) Downloads (0) Wiki (1) Graphs Branch: master

Switch Branches (4) Switch Tags (0) Branch List

AllegroGraph Python client
<http://opensource.franz.com>

HTTP Git Read-Only <http://github.com/franzinc/agraph-python.git> This URL has Read-Only access

AG4 Python tutorial update.



BruceDClayton (author)
February 04, 2010

commit 43a3da501611f9a9ecba
tree 94b2b808e6e169fef9ce
parent d76bacd8e649d0ba3e79

agraph-python / stress / events / events

100755 | 864 lines (708 sloc) | 30.155 kb raw blame history

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 ##### BEGIN LICENSE BLOCK #####
```

```

49 from franz.openrdf.repository.repository import Repository
50 from franz.openrdf.vocabulary import XMLSchema, RDF
51 from franz.openrdf.model import Literal, Statement, URI, ValueFactory
52
53 class Defaults:
54     # The namespace
55     NS = 'http://franz.com/events#';
56
57     # Number of worker processes
58     LOAD_WORKERS = 20
59     QUERY_WORKERS = 8
60
61     # Time to run queries in minutes
62     QUERY_TIME = 10
63
64     # Size of the Events
65     EVENT_SIZE = 50
66
67     # Events per commit in bulk load phase
68     BULK_EVENTS = 250
69
70     # Goal store size
71     SIZE = (10**9)
72
73     # The catalog name
74     CATALOG = 'tests'
75
76     # The repository name
77     REPOSITORY = 'events_test'
78
79     # The starting phase
80     PHASE = 1
81
82     # OPEN OR RENEW
83     OPEN = False
84
85     # The program options
86     OPT = None
87
88     LOCALHOST = 'localhost'
89     AG_HOST = os.environ.get('AGRAPH_HOST', LOCALHOST)
90     AG_PORT = int(os.environ.get('AGRAPH_PORT', '10035'))
91     AG_USER = os.environ.get('AGRAPH_USER', 'test')
92     AG_PASSWORD = os.environ.get('AGRAPH_PASSWORD', 'xyzyz')
93     PROG = sys.argv[0]
94

```

```

195 def random_origin():
196     fn = random_origin
197     return Literal(random.choice(fn.origins)).toNTriples()
198
199 random_origin.origins = ('Call Center', 'Sales', 'Front Desk' )
200
201 def random_payoption():
202     fn = random_payoption
203     return Literal(random.choice(fn.options)).toNTriples()
204
205 random_payoption.options = ('Cash', 'Credit', 'Money Order')
206
207 def random_delivery():
208     fn = random_delivery
209     return Literal(random.choice(fn.types)).toNTriples()
210
211 random_delivery.types = ('Mail', 'FedEx', 'UPS', 'Truck Freight')
212
213 def random_money():
214     return Literal(round(random.uniform(0.01, 10000.00), 2)).toNTriples()
215
216 def random_uri(prefix, limit):
217     return URI(namespace=OPT.NS,
218               localname='%s-%d' % (prefix, random.randrange(int(limit)))).toNTriples()
219
220 def random_customer():
221     # Striving for 1000 ids per prefix (e.g. customer) when SIZE is 1 billion,
222     # so by default there are SIZE/1000 ids
223     return random_uri('Customer', OPT.SIZE/1000)
224
225 def random_account():
226     return random_uri('Account', OPT.SIZE/1000)
227
228 def random_action():
229     fn = random_action
230     return Literal(random.choice(fn.actions)).toNTriples()
231
232 random_action.actions = ('Add', 'Modify')
233
234 def random_handling():
235     fn = random_handling
236     return Literal(random.choice(fn.handling)).toNTriples()
237
238 random_handling.handling = ('Nothing', 'Past Due Notice', 'Collections')
239
240 def type_uri(label):
241     return URI(namespace=OPT.NS, localname=label).toNTriples()

```

```
242
243 # The list of events to generate
244 EVENTS = None
245
246 def initialize_events():
247     global EVENTS
248
249     interaction = [
250         PredInfo(RDF.TYPE, lambda: type_uri('CustomerInteraction')),
251         PredInfo('EventTimeStamp', random_datetime),
252         PredInfo('EntityId', random_int),
253         PredInfo('OriginatingSystem', lambda: '"CRM"'),
254         PredInfo('Agent', random_name),
255         PredInfo('Direction', random_direction),
256         PredInfo('DoneInOne', random_bool),
257         PredInfo('EndDate', random_datetime),
258         PredInfo('FeeBased', random_bool),
259         PredInfo('InteractionId', random_int),
260         PredInfo('Origin', random_origin),
261         PredInfo('PayOption', random_payoption),
262         PredInfo('ReasonLevel1', random_int),
263         PredInfo('ReasonLevel2', random_int),
264         PredInfo('OriginatingSystem', random_int),
265         PredInfo('Result', random_int)]
266
267     invoice = [
268         PredInfo(RDF.TYPE, lambda: type_uri('Invoice')),
269         PredInfo('EventTimeStamp', random_datetime),
270         PredInfo('AccountId', random_account),
271         PredInfo('OriginatingSystem', lambda: '"Invoicing"'),
272         PredInfo('DueDate', random_datetime),
273         PredInfo('Action', random_action),
274         PredInfo('TotalAmountDue', random_money),
275         PredInfo('AmountDueHandling', random_handling),
276         PredInfo('LegalInvoiceNumber', random_int),
277         PredInfo('PreviousBalanceAmount', random_money),
278         PredInfo('TotalFinanceActivites', random_money),
279         PredInfo('BillDate', random_date),
280         PredInfo('TotalUsageCharges', random_money),
281         PredInfo('TotalRecurringCharges', random_money),
282         PredInfo('TotalOneTimeCharges', random_money)]
283
284     payment = [
285         PredInfo(RDF.TYPE, lambda: type_uri('AccountPayment')),
286         PredInfo('EventTimeStamp', random_datetime),
287         PredInfo('AccountId', random_account),
```

Testing with 25 loading, 16 querying processes. Repository contains 0 triples.

Phase 1: Baseline 50 triple commits.

10,000,000 total triples processed in 432.464979 seconds (23,123.259656 triples/second, 462.465193 commits/second). Store contains 10,000,000 triples.

Phase 2: Grow store to about 100,000,000 triples.

90,000,000 total triples processed in 2,452.194867 seconds (36,701.814039 triples/second, 2.936145 commits/second). Store contains 100,000,000 triples.

Phase 3: Perform 50 triple commits.

10,000,000 total triples processed in 571.308110 seconds (17,503.689909 triples/second, 350.073798 commits/second). Store contains 110,000,000 triples.

Phase 4: Perform customer/date range queries with 16 processes for 1 minute.

1,001,800 total triples returned over 2,771 queries in 59.155107 seconds (16,935.139677 triples/second, 46.842955 queries/second, 361.000000 triples/query).

Phase 5: Shrink store by 1 month.

-10,000,000 total triples processed in 38.246072 seconds (-261,464.760309 triples/second, 20.917181 commits/second). Store



The next challenge

- Keep the 4.0 features in a trillion triples environment
- I'd like to report on that at the next workshop

thanks

**TopBraid
Composer
Ontology
Modeling**

**RacerPro
OWL Description
Logic**

**AGWebview
RDF Browser
Server**

**Gruff
RDF Browser**

**Pepito
Data Mining**

Java: Sesame / Jena Python C# Ruby Lisp Perl

REST Client / Server Protocol

SPARQL

Prolog

RDFS++

GeoTemporal Reasoning / Social Network Analytics

Storage / Transactions / Federation

Oracle
Postgres
MySQL

