Exelixi

Neil Nistler

Exelixi

- Exelixi is a distributed framework based on Apache Mesos. It is mostly implemented in Python using gevent for high performance concurrency. It is intended to run cluster computing jobs in pure Python. By default it runs GA's (genetic algorithms) at scale. However it can handle a broad range of other problem domains.

Genetic Algorithms

- In general a GA is a search heuristic that mimics the process of natural selection in biological evolution. This approach is used to generate candidate solutions for optimization and search problems, especially in cases where the parameter space is large and complex.

Genetic Algorithms

- In a GA, a population of candidate solutions called “individuals” to an optimization problem gets evolved toward improving solutions. Each candidate solution has a feature set called its “chromosomes” which can be altered and recombined. The fitness for each individual gets evaluated or approximated using a fitness function applied to its feature set.

Fitness Function

Randomly choose initial population

while not (termination criteria):

    evaluate each individual's fitness function

    prune population, retaining best-fit individuals

    randomly select pairs to mate

    replenish population

        apply crossover operator

        apply mutation operator

report results

Genetic Algorithms

- Evolution starts with a set of randomly generated individuals, then iterates through successive generations. A stochastic process called selection preserves the individuals with better fitness as parents in the next generation, and gets rid of the rest. Some individuals get altered, based on a mutation operation. Pairs of parents are selected at random with replacement from the population, then mated to produce new individuals based on a crossover operation.

Genetic Algorithms

The algorithm terminates when the population reaches some user-defined condition for example:

- Acceptable fitness for some individual
- Threshold aggregate error for the population overall
- Maximum number of generations iterated
- Maximum number of individuals evaluated

System Dependencies

- Apache Mesos 0.15.0 rc4
- Git
- Python version 2.7 with Anaconda as the recommended platform, plus the following libraries:
    - setuptools
    - dev
    - pip
    - protobuf
    - gevent
    - psutil
    - cython
    - hat-trie

System Dependencies

- Much of this framework was developed for Ubuntu 13.10 release.
- The installer script bin/local_install.sh uses aptitude, which you may need to rework for any OS other than Debian or Ubuntu.

Getting Started!

- The following instructions are based on using the Elastic Mesos service, which uses Ubuntu Linux servers running on Amazon AWS.

- First launch an Apache Mesos cluster. Once you have confirmation that your cluster is running, then use ssh to login on any of the Apache Mesos masters:

  - ssh -A -l ubuntu <master-public-ip>

Getting Started (2)

- You must use the Python bindings for Apache Mesos, by installing the Python egg for that exact release.

- On the master, download the master branch of the Exelixi code repo on GitHub and install the required libraries:

  - wget https://github.com/ceteri/exelixi/archive/master.zip ; \

    unzip master.zip ; \

    cd exelixi-master ; \

    ./bin/local_install.sh

Getting Started (3)

You can test the installation at any point by attempting to import the mesos package into a Python REPL:

```
python -c 'import mesos'
```

If there is no ImportError exception thrown, then your installation should be complete.

Getting Started (4)

- Now, build a tarball/container to distribute the Exelixi code to each of the Apache Mesos slaves.
- If you have customized the code by forking your own GitHub code repo, then substitute that download URL instead.
- If you have customized by subclassing the uow.UnitOfWorkFactory default GA, then place that Python source file into the src/ subdirectory.

Getting Started (5)

Next, run the installation command on the master, to set up each of the slaves.

- ./src/exelixi.py -n localhost:5050 | ./bin/install.sh

Now, launch the framework, which in turn launches the worker services remotely on slave nodes. In this case it runs workers on 2 slave nodes:

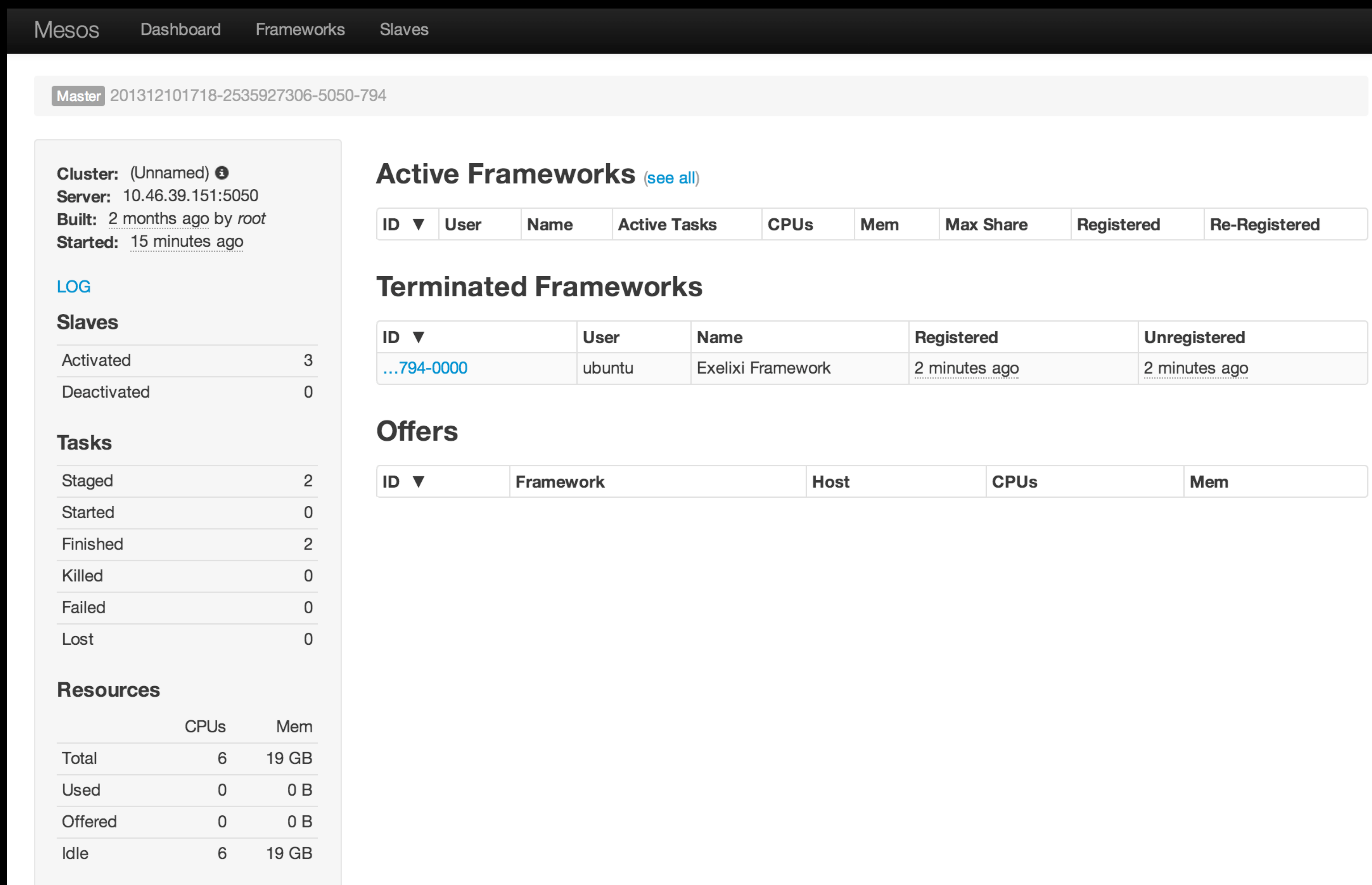
- ./src/exelixi.py -m localhost:5050 -w 2

Getting Started (6)

- Once everything has been set up successfully, the log file in exelixi.log will show:
  - all worker services launched and init tasks completed

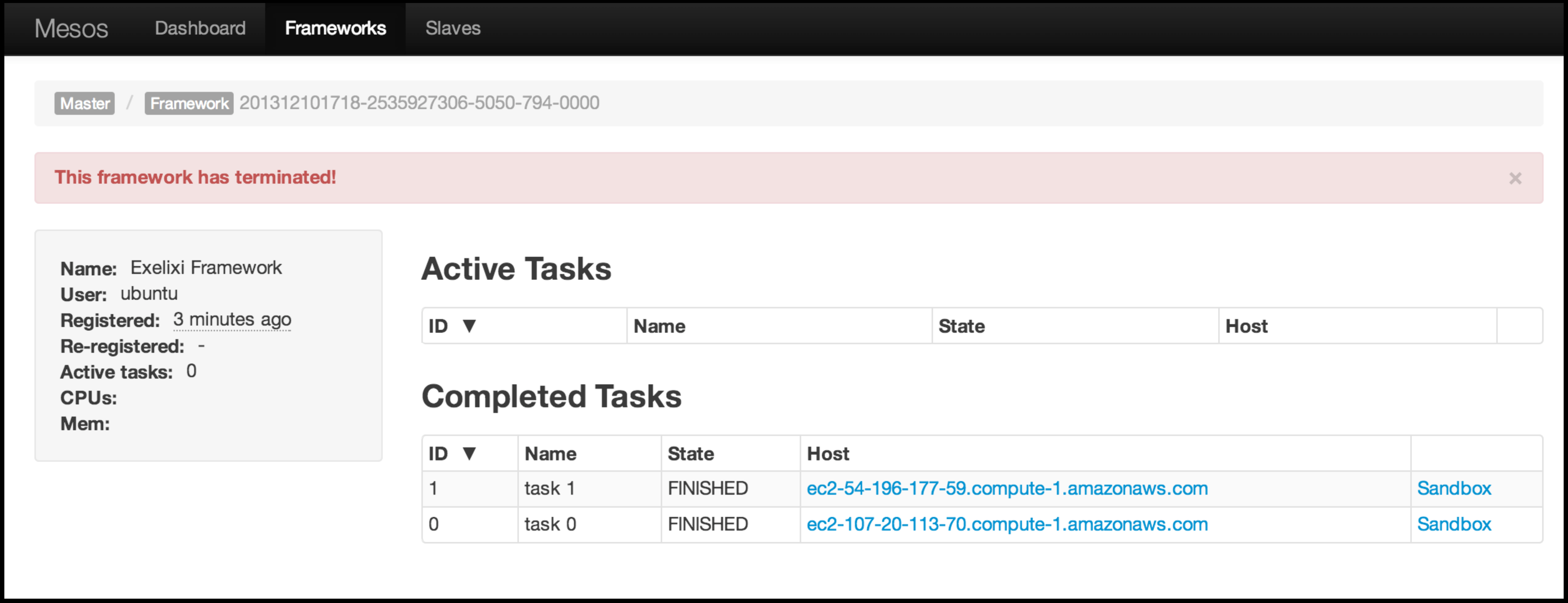
Exelixi

Once the Exelixi framework is running, it will show up as an entry “Exelixi Framework” in the Mesos console:



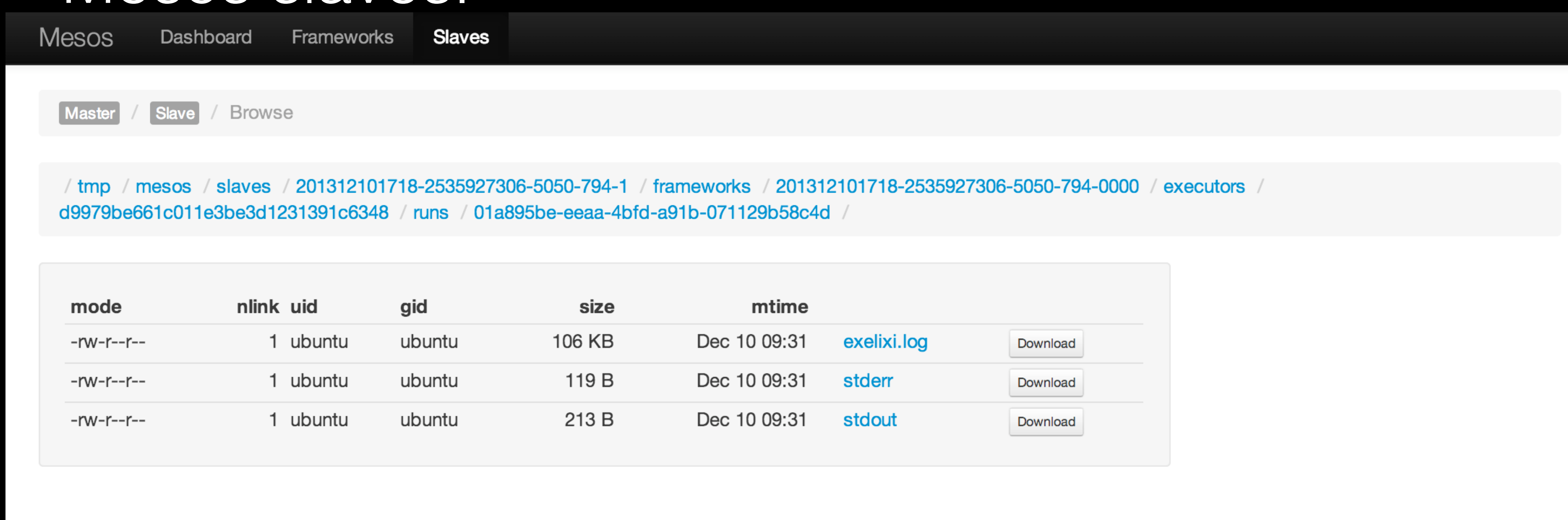
Exelixi

- Click on the framework ID to list the tasks scheduled through Mesos. There will be one for each worker.



Exelixi

- Now, you can click on the Sandbox link for one of the tasks.
- From here you can click on the sandboxed files captured for that specific task.
- This is useful for debugging code running on the Mesos slaves.



TSP Example

- First define a constructor with the GA settings

```
class TSPFactory (UnitOfWorkFactory):
    # UnitOfWork definition for Traveling Salesperson Problem

    def __init__ (self):
        self.n_pop = 10
        self.n_gen = 23
        self.max_indiv = 2000
        self.selection_rate = 0.2
        self.mutation_rate = 0.02
        self.term_limit = 5.0e-03
        self.hist_granularity = 3
```

TSP Example

In the previous slide we see that:

- we start with a population size of 10
- We run for a maximum of 23 generations
- We have a limit of 2000 individuals
- We use 0.2 for the selection rate
- We use 0.02 for the mutation rate
- The self.term_limit and self.hist_granularity are really for much larger scale problem

TSP Example

- self.route_meta and self.route_cost

```
# cost matrix for an example TSP: optimize the route
# tuple definition: (name, addr, duration)

self.route_meta = ( ( "Home", "secret", 0 ),
                    ( "Piazzas Fine Foods", "3922 Middlefield Rd, Palo Alto, CA 94303", 45 ),
                    ( "Mountain View Public Library", "585 Franklin St, Mountain View, CA 94041", 30 ),
                    ( "Seascapes Fish & Pets Inc", "298 Castro St, Mountain View, CA 94041", 10 ),
                    ( "Dana Street Roasting Company", "744 W Dana St, Mountain View, CA 94041", 20 ),
                    ( "Supercuts", "2420 Charleston Rd, Mountain View, CA 94043", 60 ),
                  )

self.route_cost = ( ( 0, 7, 11, 12, 14, 8 ),
                    ( 7, 0, 18, 18, 19, 5 ),
                    ( 14, 19, 0, 2, 3, 19 ),
                    ( 12, 20, 3, 0, 1, 19 ),
                    ( 12, 18, 3, 1, 0, 18 ),
                    ( 8, 5, 18, 18, 19, 0 ),
                  )

# sampling parameters
self.length = len(self.route_cost) - 1
self.min = 1
self.max = self.length
```

TSP Example

For this example the optimal route must begin and end at Home.

In the previous slide we see:

- self.route_meta includes a list of stops that must be made.

- self.route_cost is a matrix that denotes the time in minutes to get from a given starting point to all the other locations.

TSP Example

- One of the most important aspects of machine learning is effective representation for a given problem

- When you extend Exelixi to specify a different UnitOfWorkFactory, you need to rework the generate_features() method to handle representation for the problem being solved

- In this case, the feature_set is just a list of integers, a sequence of points based on the rows in the cost matrix

```
def generate_features (self):
    # generate a new feature set for TSP
    features = []
    expected = list(xrange(self.min, self.max + 1))

    # sample row indices in the cost matrix, without replacement
    for _ in xrange(self.length):
        x = sample(expected, 1)[0]
        features.append(x)
        expected.remove(x)

    return features
```

TSP Example

For evaluating TSP, we calculate fitness based on two criteria

- Did each of the required points get visited on the candidate route
- Does the candidate route minimize travel time

For the first question, a set of differences is a good estimator

For the second question, a tally of the time required for the route, then normalize based on the worst case scenario of the TSP to go back and forth from his home to each point in the route.

TSP Example

```
def get_fitness (self, feature_set):
    # Determine the fitness ranging [0.0, 1.0]; higher is better

    # 1st estimator: all points were visited?
    expected = set(xrange(self.min, self.max + 1))
    observed = set(feature_set)
    cost1 = len(expected - observed) / float(len(expected))

    # 2nd estimator: travel time was minimized?
    total_cost = 0
    worst_case = float(sum(self.route_cost[0])) * 2.0
    x0 = 0

    for x1 in feature_set:
        total_cost += self.route_cost[x0][x1]
        x0 = x1

    total_cost += self.route_cost[x0][0]
    cost2 = min(1.0, total_cost / worst_case)

    # combine the two estimators into a fitness score
    fitness = 1.0 - (cost1 + cost2) / 2.0

    if cost1 > 0.0:
        fitness /= 2.0

    return fitness
```

TSP Example

- In the previous slide it is important to notice that if a candidate solution skips some of the required points, then its fitness score is penalized (divided by 2). Bad solutions are not set to zero. When creating a fitness function for a GA, there is no bad solutions, except for no solution. Even bad solutions contain some information. Even if it’s only a starting point from which to evolve better solutions.

TSP Example

- Using a GA to solve the TSP problem gives us the following results:

gen 0	size	10	total 10	mse	1.13e-01	max	7.69e-01	med	6.91e-01	avg	5.28e-01
gen 1	size	10	total 17	mse	2.58e-01	max	7.69e-01	med	6.30e-01	avg	3.13e-01
gen 2	size	10	total 23	mse	2.38e-01	max	7.69e-01	med	6.30e-01	avg	3.67e-01
gen 3	size	10	total 29	mse	8.26e-02	max	7.69e-01	med	7.60e-01	avg	5.65e-01
gen 4	size	10	total 35	mse	1.58e-01	max	7.69e-01	med	7.64e-01	avg	4.14e-01
gen 5	size	9	total 39	mse	7.15e-02	max	7.79e-01	med	7.71e-01	avg	4.82e-01
gen 6	size	9	total 42	mse	8.50e-02	max	7.79e-01	med	7.71e-01	avg	4.64e-01
gen 7	size	10	total 46	mse	1.14e-01	max	7.79e-01	med	7.71e-01	avg	3.90e-01
gen 8	size	10	total 49	mse	1.28e-01	max	7.79e-01	med	7.71e-01	avg	3.68e-01
gen 9	size	9	total 51	mse	9.71e-02	max	7.84e-01	med	7.71e-01	avg	3.80e-01
gen 10	size	10	total 54	mse	8.68e-02	max	7.84e-01	med	7.71e-01	avg	3.49e-01
gen 11	size	10	total 57	mse	6.79e-02	max	7.84e-01	med	7.70e-01	avg	3.64e-01
gen 12	size	9	total 62	mse	1.15e-01	max	7.84e-01	med	7.67e-01	avg	5.28e-01
gen 13	size	7	total 65	mse	1.77e-01	max	7.84e-01	med	7.77e-01	avg	5.16e-01
gen 14	size	9	total 70	mse	6.41e-02	max	7.84e-01	med	7.69e-01	avg	5.82e-01
gen 15	size	8	total 74	mse	8.55e-02	max	7.84e-01	med	7.74e-01	avg	6.20e-01
gen 16	size	9	total 79	mse	1.03e-01	max	7.84e-01	med	7.24e-01	avg	6.06e-01
gen 17	size	6	total 81	mse	7.82e-02	max	7.84e-01	med	7.79e-01	avg	4.94e-01
gen 18	size	5	total 82	mse	6.57e-02	max	7.84e-01	med	7.81e-01	avg	5.94e-01
gen 19	size	7	total 86	mse	8.43e-02	max	7.84e-01	med	7.76e-01	avg	6.07e-01
gen 20	size	7	total 89	mse	7.56e-02	max	7.84e-01	med	7.76e-01	avg	6.22e-01
gen 21	size	7	total 92	mse	6.77e-02	max	7.84e-01	med	7.76e-01	avg	5.38e-01
gen 22	size	6	total 94	mse	5.98e-02	max	7.84e-01	med	7.79e-01	avg	6.29e-01
indiv	0.7837	8	[2, 3, 4, 1, 5]								
indiv	0.7788	4	[1, 5, 2, 4, 3]								
indiv	0.7788	10	[2, 4, 3, 5, 1]								
indiv	0.7740	21	[3, 4, 2, 5, 1]								

TSP Example

- From the results we see that the route [2, 3, 4, 1, 5] is the best among all the possible routes for the TSP. This route was given a fitness score of 0.7837.

- From the TSP's home they would first go to the library, then to the pet store, then to the coffee shop, then to Piazzas, and finally to the Supercuts.

TSP Analysis

- The best route was found during generation 8 of this GA run.
- A total of 49 candidate solutions had been evaluated by that stage.
- Given the complexity of the TSP problem, and a route size of $n=6$, then $n! = 720$ would be the expected upper bound for complexity.
- $49 < 720$, shows that the GA calculated a solution efficiently.

Where To Go From Here

- This really has been just an introduction to Exelixi and it's capabilities and only scratches the surface.
- For more information on Apache Mesos please visit
- mesos.apache.org
- For more information on genetic algorithms and there practical uses please visit
- http://en.wikipedia.org/wiki/Genetic_algorithm
- For more in-depth information on Exelixi and Orchestrating a Unit of Work please visit
- https://github.com/ceteri/exelixi/wiki

Resources

- https://github.com/ceteri/exelixi/wiki
- http://en.wikipedia.org/wiki/Genetic_algorithm
- mesos.apache.org