

Supplementary Material for:

Processing shotgun proteomics data on the Amazon Cloud with the Trans-Proteomic Pipeline

Joseph Slagel, Luis Mendoza, David Shteynberg, Eric W. Deutsch* and Robert L. Moritz

SS1: EC2 Provisioning in amztp

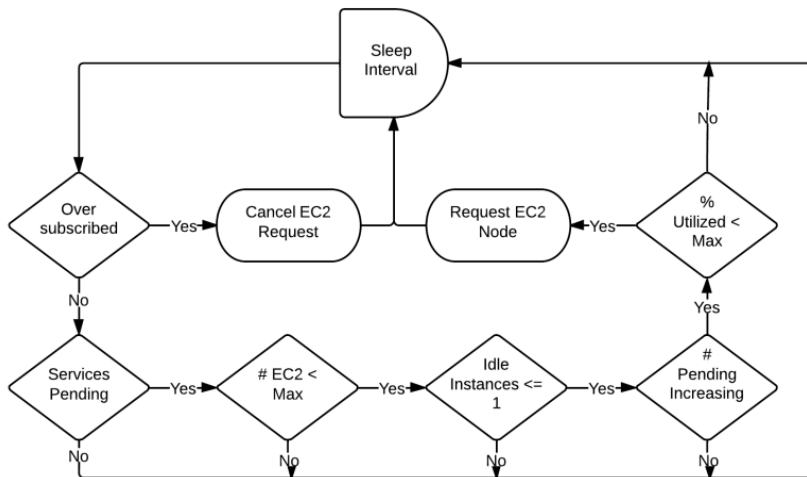
One of the commonly overlooked aspects to cloud computing is how best to provision EC2 instances when processing data on the cloud. In general most cloud applications simply resort to requiring the user to specify the number of EC2 instances to instantiate when the application is initiated then allow users to dynamically either allocate additional or terminate existing instances based on their observations. So in the simplest scenario when a user has 100 MS searches to execute and they intend to minimize the search time they would instantiate 100 EC2 instances. The amztp program follows this same precedence by providing the "launch" and "terminate" commands and options such as "-ec2-num <x>" for specifying the number of nodes to operate on. And while this approach can in most scenarios minimize search time it does this by not considering the cost. To efficiently manage cost one has to consider the many other cloud properties such as:

- EC2 instances are billed per hour regardless of how much time is actually used.
- Data transfer rates that are affected by bandwidth limitations and flux at the client, Internet and Amazon.
- Large variance in the execution time of different search programs and different search inputs.
- Infrequent errors necessitating retries on data transfers or search executions.
- The availability of the instances due to market price if using EC2 spot instances.

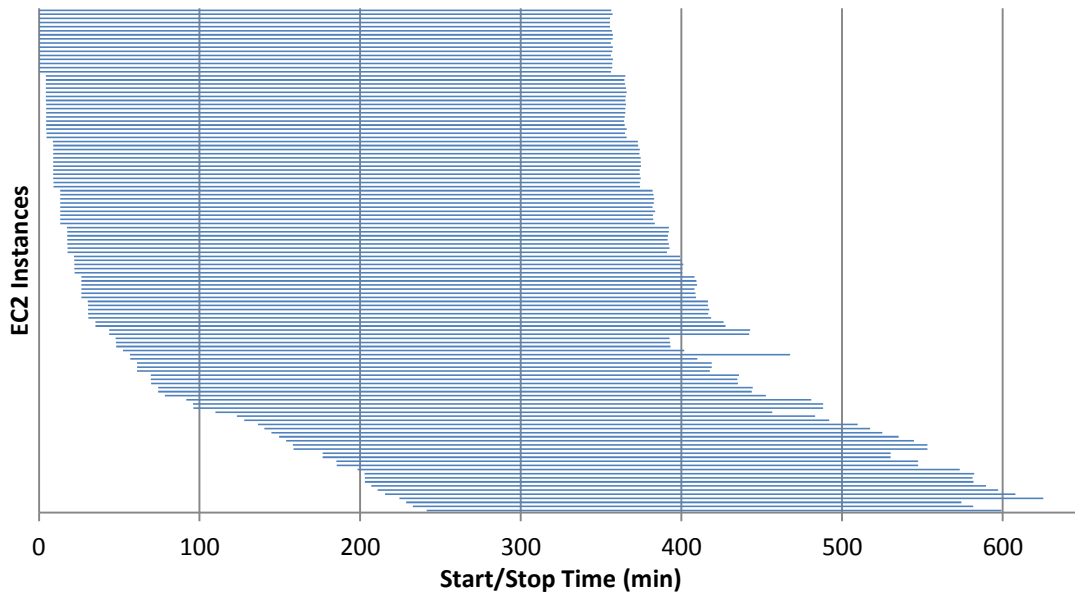
These factors affecting cost are particularly noticeable when large data sets are processed as the effect of the idle instances, substandard job scheduling, and data transfer rates can significantly increase the program's usage of AWS resources and subsequently increase overall AWS charges.

To address the issue of cost efficiency a minimal EC2 provisioning algorithm was implemented in the amztp program. This algorithm is invoked periodically by a background process started by *amztp* (which is also responsible for performing file uploads, downloads, and monitoring). Supplementary Figure 1 shows the decision tree that is used for the determination. Instantiating new EC2 instances is based on conditions including: pending jobs, maximum number of instances allowed, number of instances that are idle, and the number of pending is greater than the number started. Several additional provisions were added to limit the rate at which instances are launched. The

first requires that the rate of pending jobs is increasing over time which implies that the jobs are being uploaded faster than being completed. The other limits the number of running instances to a configurable percentage of the number of pending jobs in order to not oversubscribe the number of instances. The decision to terminate EC2 instances is left up to the instances themselves which will shut down if no new jobs are available within the last two minutes of the current billing hour. The presumption here is that once an instance is started you might as well use the full hour already billed for. However it will cancel EC2 instance requests if they haven't yet been fulfilled and the pending jobs are trending downward under the presumption that the current work load is being handled.



Supplementary Figure 1: EC2 Provisioning decision tree



Supplementary Figure 2: Depiction of the active nodes during a search of many files with many instances, in this case the search described in section SS6 below. Most node lifetimes are just under an integer hour since they are programmed to shut down shortly before they would incur a charge for another hour.

Supplementary SS2: Enhancements to TPP's Petunia Web based Graphical User Interface to Support Cloud Computing

Support for invoking the *amztp* command line tool has been added to the Trans-Proteomic Pipeline's graphical user interface Petunia. These enhancements include:

- AWS console report, shown in Supplementary Figure 3
- Enhanced jobs report page showing AWS and local jobs, as shown in Supplementary Figure 4
- Ability to choose location where searches are executed (local vs. cloud), as shown in Supplementary Figure 5
- Interface to the *s3cmd* for syncing the contents of a local folder with a objects in Amazon's Simple Storage Solution (S3), shown in Supplementary Figure 6

The screenshot displays the Petunia GUI's AWS console interface. At the top, there are three browser tabs for 'ISB/SPC Trans Proteomic'. The address bar shows 'localhost/tpp-bin/tpp_gui.pl?Action=display&page=clusters'. The main header is 'ISB/SPC Trans Proteomic Pipeline - clusters'. Below this is a navigation bar with tabs: 'Home', 'Manage My Account', 'Manage Session(s)', and 'Amazon Cloud'. A status bar indicates 'You are logged in as guest' with a 'Log Out' button. The 'Amazon Cloud' tab is active, showing a 'Register Amazon EC2 Account' button and a 'Registered Amazon Account Status' section. The status section shows the account is running as of 'Fri Sep 27 15:48:19 2013' with a client background process 'running (pid=5812)'. It contains a table of EC2 instances with 2 running instances. Below this is an 'SQS Queue Name' table with 4 queues. At the bottom is an 'S3 File Name' table with 7 files. At the very bottom, there are buttons for 'De-register this Amazon EC2 Account' and 'Shut down instances and delete all data'.

Registered Amazon Account Status [Show / Hide]

Status as of Fri Sep 27 15:48:19 2013
Client background process is: running (pid=5812)

EC2	Instance	Status	AMI_id	Type	Started	Public_DNS	TPP_log	Server_log
	i-87b76fb3	pending	ami-4fb0227f	m1.xlarge	2013-09-27 22:48:09.000Z	ec2-54-202-88-254.us-west-2.compute.amazonaws.com	TPP_log	server_log
	i-40af2174	running	ami-4fb0227f	m1.xlarge	2013-09-27 22:46:22.000Z	ec2-54-214-165-203.us-west-2.compute.amazonaws.com	TPP_log	server_log

Found 2 instances

SQS	Queue_Name	Messages	Timeout
	AMZTPP-AKIAIF6WUN6ELKCCONCQ-done	1 / 0	600
	AMZTPP-AKIAIF6WUN6ELKCCONCQ-download	0 / 0	43200
	AMZTPP-AKIAIF6WUN6ELKCCONCQ-service	2 / 0	300
	AMZTPP-AKIAIF6WUN6ELKCCONCQ-upload	2 / 1	43200

Found 4 queues

S3	File_Name	Size	Last_Modified
	C:/inetpub/wwwroot/ISB/data/21D1-Exosome/CanineMerged-CRap-Decoy-1.fasta	31.7 MB	2013-09-27 21:26:18.000Z
	C:/inetpub/wwwroot/ISB/data/21D1-Exosome/O090930-01.mzML.gz	14.6 MB	2013-09-27 21:31:04.000Z
	C:/inetpub/wwwroot/ISB/data/21D1-Exosome/O090930-03.mzML.gz	14 MB	2013-09-27 22:46:19.000Z
	C:/inetpub/wwwroot/ISB/data/21D1-Exosome/tandem.amztpd.params	5.1 KB	2013-09-27 21:33:15.000Z
	C:/inetpub/wwwroot/ISB/data/21D1-Exosome/taxonomy.xml	233 B	2013-09-27 21:33:16.000Z
	C:/inetpub/wwwroot/ISB/data/parameters/isb_default_input_kscore.xml	9.7 KB	2013-09-27 21:31:02.000Z
	var/log/i-60b66e54-amztpd.log	8 KB	2013-09-27 22:32:18.000Z

Found 7 files

De-register this Amazon EC2 Account Shut down instances and delete all data

Supplementary Figure 3: Screenshot of the main AWS console (Amazon Cloud tab) in the Petunia GUI. The three data sections list the number of running instances (2 in this case), the number of SQS messages of the 4 classes, and the files currently stored in Amazon’s Simple Storage Service S3. This AWS console enables users to see an overview of the status of their Amazon virtual cluster and stop its operation.

ISB/SPC Trans Proteomic Pipeline - jobs

Home | Account | Pre-Process | mzXML Utils | Analysis Pipeline (Tandem) | Decoy | Utilities | SpectraST Tools | Jobs You are logged in as guest. [Log Out](#)




Home **Jobs**

Commands Status [Show / Hide]

Status as of: Fri Sep 27 15:46:28 2013

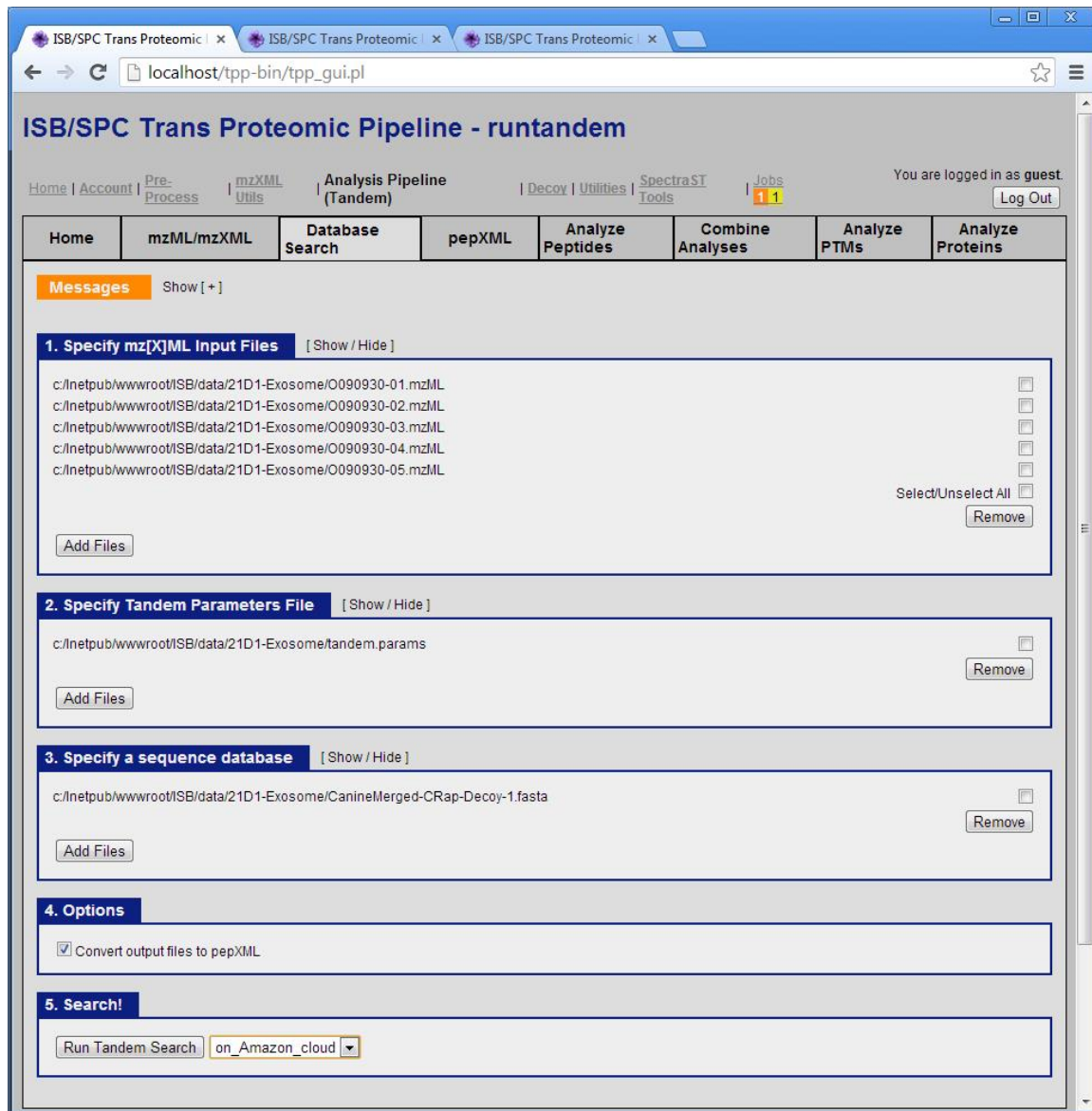
All Jobs [Show / Hide]

Session ID	Job	Location	Start date / time	Actions	Status	Output
T238VZFAP	runtandem	Amazon cloud	2013-09-27 15:46:07		queued	View
T238VZFAP	msconvert	localhost	2013-09-27 15:45:04	Delete Log	finished	View
T238VZFAP	runtandem	Amazon cloud	2013-09-27 14:26:07		queued	View

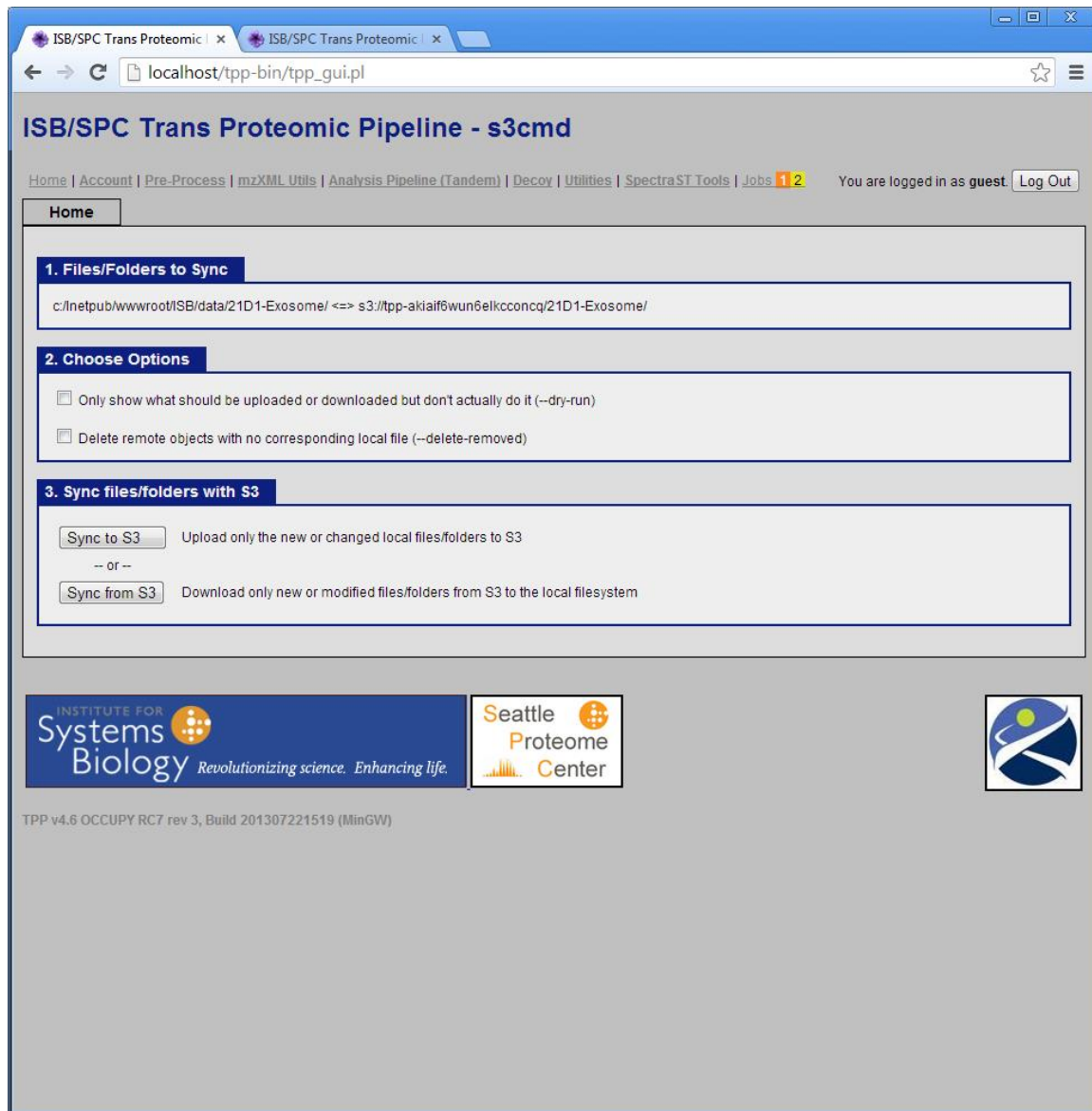




TPP v4.6 OCCUPY RC7 rev 3, Build 201307221519 (MinGW)

Supplementary Figure 4: Screenshot of the main jobs console in the Petunia GUI. This view shows one completed data conversion job, which had been run locally, and two different queued X!Tandem search jobs, which are running on the Amazon Cloud.



Supplementary Figure 5: Screenshot of the database search launching page in the Petunia GUI. This view shows a search job for five mzML files using X!Tandem about to be submitted to the Amazon Cloud. The list box at the bottom currently allows users to choose to run X!Tandem locally or on a configured AWS account. Future enhancements could allow submissions to a locally configured compute cluster or other cloud computing frameworks.



Supplemental Figure 6: Screenshot of interface to s3cmd for syncing the contents of a folder with Amazon's Simple Storage Service (S3).

SS3: Comparison of different EC2 instance types

Amazon's EC2 provides a wide variety of different instance types from which to choose with different memory, disk storage, CPU, and networking capacity. Some of these are summarized in Supplementary Table 1. Accurately comparing the performance of each EC2 type can be difficult, since while Amazon provides specifications for each EC2 type, these are **minimum** expected specifications. Studies have shown that actual performance measured between instances of the same EC2 type can vary as much as 60%. This variance is largely due to the different physical hardware configurations used to provide for the same types [1]. In order to evaluate the approximate performance of each EC2 types as related to proteomics needs, a representative mzML file containing 3,831 MS/MS spectra generated on a LTQ-Velos Orbitrap was searched with four of the supported search programs in triplicate on each of the EC2 types shown in Supplementary Table 2. We present the results and the average search time of the three runs for each search engine on the different EC2 types. These data are also presented in Supplementary Figure 7 to illustrate cost versus time where cumulative time is the sum of the averages of all four searches and cost is calculated using the price per minute for the EC2 type (disregarding that EC2 instances are paid in increments of 1 hour).

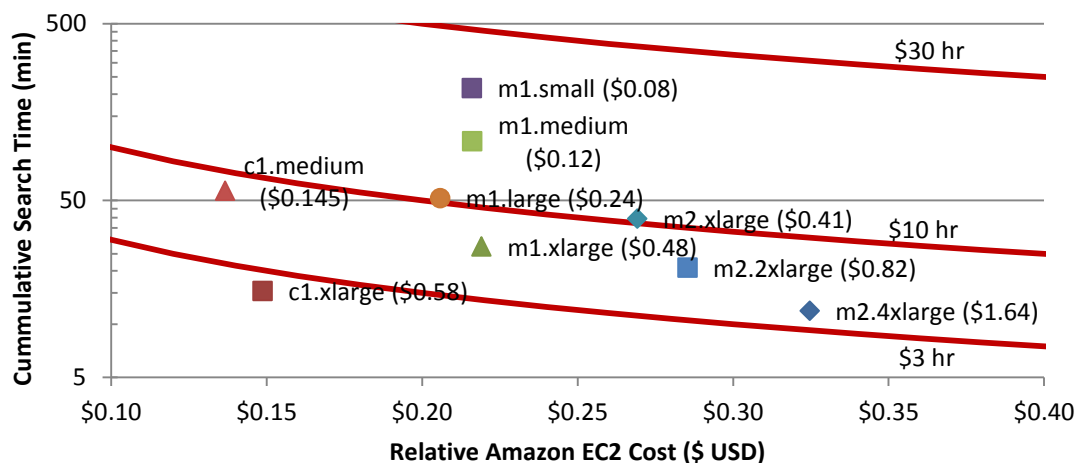
The results show that an AWS c1.xlarge instance type with 20 equivalent cores is both the fastest and most efficient as measured by unit time per unit cost, approximately 20% more efficient than m1.xlarge and 60% more efficient than m1.large. The relative efficiency will, of course, vary somewhat by dataset and search engine. For the fastest EC2 types, m2.4xlarge was on average 25% faster than the c1.xlarge, but the cost difference was 3 times more expensive per hour. Similarly the c1.medium and m1.large timings were within 10% but the cost difference was over 50%. It is important to point out that AWS bills usage per unit hour on all EC2 types so cost savings will be affected by the fraction of the final hour that is not used for processing.

Supplementary Table 1: Partial List of Amazon EC2 Types

EC2 Type	CPU Description	Memory	Cost/Hr²
m1.small	1 virtual cores 1 EC2 units¹	1.7 GiB	\$ 0.060
m1.medium	1 virtual cores 2 EC2 units¹	3.75 GiB	\$ 0.120
c1.medium	2 virtual cores 2.5 EC2 units¹	1.7 GiB	\$ 0.145
m1.large	2 virtual cores 2 EC2 units¹	7.5 GiB	\$ 0.240
m2.xlarge	2 virtual cores 3.25 EC2 units¹	17.1 GiB	\$ 0.410
m1.xlarge	4 virtual cores 2 EC2 units¹	15 GiB	\$ 0.480
c1.xlarge	8 virtual cores 2.5 EC2 units¹	7 GiB	\$ 0.580
m2.2xlarge	4 virtual cores 3.25 EC2 units¹	34.2 GiB	\$ 0.820
m2.4xlarge	8 virtual cores 3.25 EC2 units¹	68.4 GiB	\$ 1.640
¹ One EC2 compute unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor ² Amazon EC2 costs for US West (Oregon) region and partial hours are billed as a full hour. Costs for other regions may vary and these prices are subject to change.			

Supplementary Table 2: Average Search Time (minutes) for one mzML file on each EC2 Type

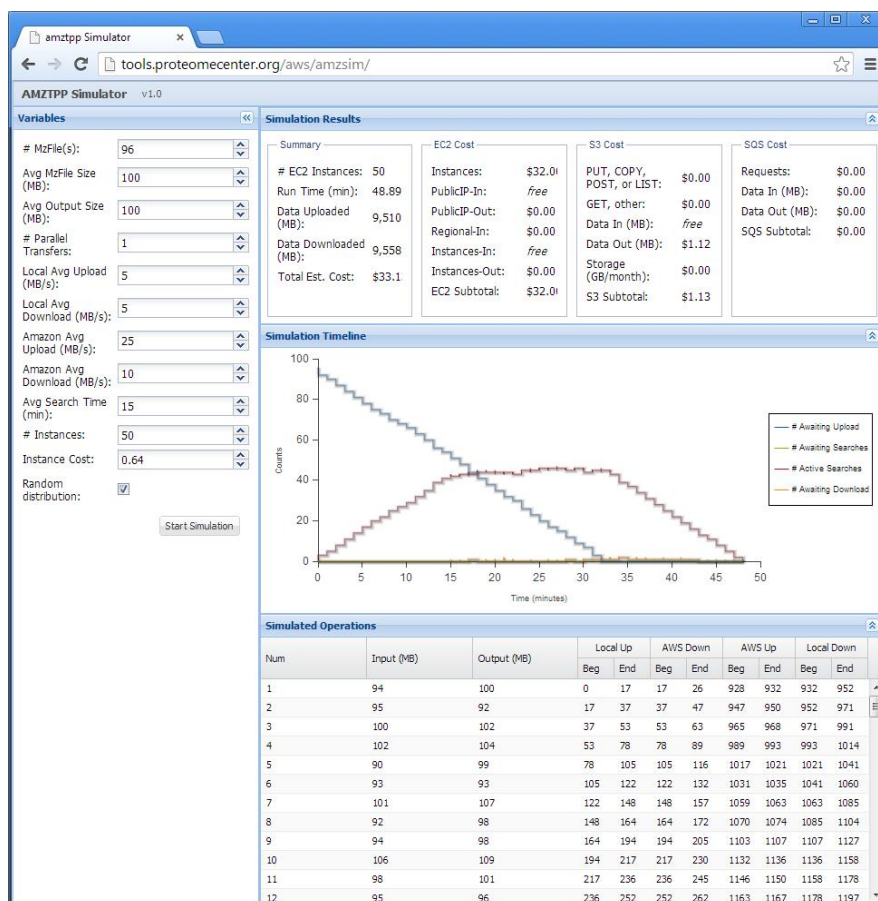
<i>EC2 Type</i>	<i>Comet</i>	<i>Myrimatch</i>	<i>OMSSA</i>	<i>X!Tandem</i>	<i>Total</i>
m1.small	98.8	88.7	11.0	17.6	216.0
m1.medium	49.5	44.0	5.5	9.0	108.1
c1.medium	26.7	21.5	3.1	5.3	56.5
m1.large	21.5	21.7	3.0	5.3	51.5
m2.xlarge	16.4	16.7	2.3	4.0	39.4
m1.xlarge	10.7	11.2	1.8	3.7	27.4
c1.xlarge	5.5	5.8	1.2	2.8	15.4
m2.2xlarge	8.2	8.5	1.4	2.8	20.9
m2.4xlarge	4.1	4.5	1.0	2.3	11.9



Supplementary Figure 7. Semilog plot of cumulative time for a representative unit of work involving the combination of running Comet, MyriMatch, OMSSA, and X!Tandem searches versus the incremental cost of searches (ignoring that EC2 instances are billed in increments of 1 hour). The result for each instance type is plotted and labeled with a few curves of constant time \times cost are overlaid. The best performance is to the lower left of the plot, where the job is completed with the smallest cost and time. The c1.medium completes the job with the lowest cost, but c1.xlarge has the optimal cost vs. time tradeoff.

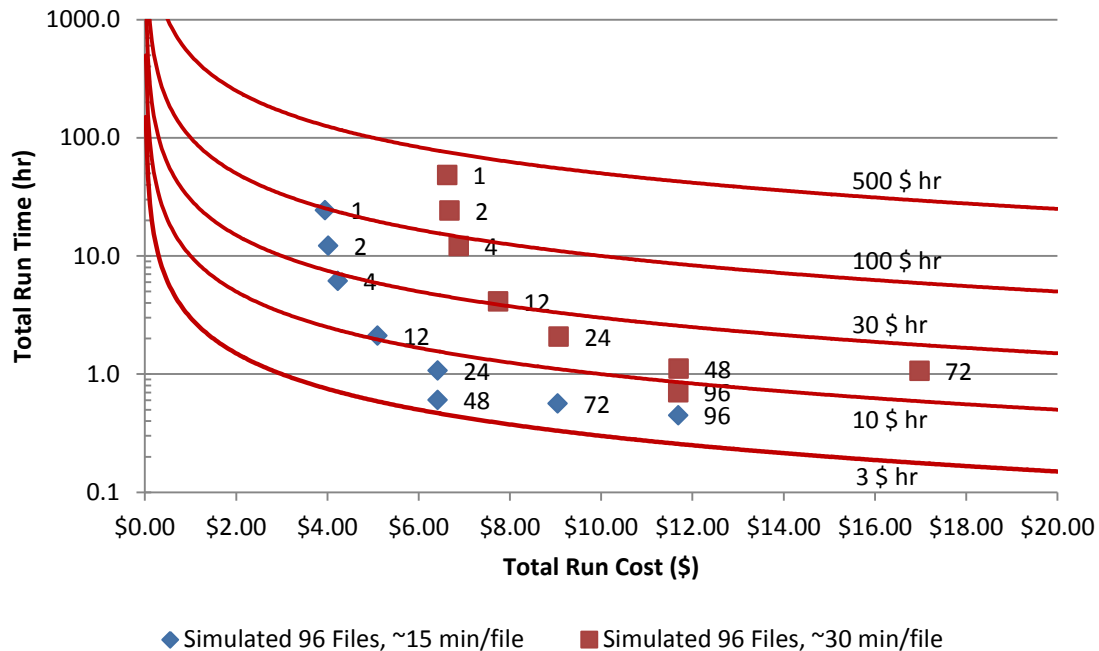
SS4: Evaluating the use of multiple EC2 instances for analysis

One advantage cloud resources affords is the ability to launch multiple EC2 instances to significantly reduce the overall analysis time when processing multiple MS/MS spectrum files. To further explore the relationship of time versus cost, a simple web based simulator, amzsim, has been developed and is freely available to the community at <http://tools.proteomecenter.org/aws/amzsim>. The simulator considers numerous parameters including the number of mzML files, the average upload/download speeds, average file sizes, and average search times. The simulated results include costs for EC2, SQS, and S3 services, a timeline of the simulated jobs, and a table containing all of the simulated data as shown in Supplementary Figure 8.





Supplementary Figure 8. Screenshot of amzsim, a web application for simulating the execution of multiple MS/MS search identification jobs in parallel on the cloud.

Based on the simulator, we show the simulation results of running 96 mzML files on a series of different EC2 instance counts with search times of 15 and 30 minutes in Supplementary Figure 9. Several curves of constant time \times cost are overlaid in red; each curve represents a constant tradeoff between cost and time (i.e. along each line a doubling of time yields a halving of cost and vice versa). The data points follow a similar pattern of marked decrease in processing time with only a modest increase in overall cost at lower node counts. As the total search time breaks below the 1 hour barrier, there are some data points with erratic costs depending on whether nodes shut down prior to the billing of the second hour increment. Data points closest to the red 3 \$ \times hr curve are the most efficient in terms of cost and time. Data points along one of the red curves would be considered equally efficient. Clearly each job (i.e. set of searches) have an optimum number of nodes if both time and cost are a factor. Fewer nodes usually complete a job with lower overall cost. Too many nodes causes too much wasted money due to idling nodes. The simulator can help estimate what the optimum number of nodes is if some basic parameters about the job are known.



Supplementary Figure 9. Semilog scatterplot showing the simulated run times versus the cost for processing a set of 96 mzML files through X!Tandem with 1, 2, 4, 12, 24, 48, 72, and 96 EC2 instances performing the work in parallel. A few curves of constant time \times cost are overlaid; each curve represents a constant tradeoff between cost and time (i.e. along each line a doubling of time yields a halving of cost and vice versa).

Clearly making temporary use of a large number of nodes is beneficial for swift analysis of data with relatively little increase in marginal cost. This is very difficult to achieve with a local computer cluster as it is very often either oversubscribed with impatient users waiting for their share of the available compute resources during work hours or running well below capacity, sometimes idle for significant periods of time such as evenings and weekends. With the cloud computing infrastructure demonstrated here, each user can have a large compute cluster of their own that only runs when there are data to be actively processed. There is no contention over CPU resources and data processing occurs quickly. The advantages and disadvantages of AWS cloud computing vs. a local compute cluster summarized in Supplementary Figure 10. Nevertheless, if a user already has free access to a local compute cluster where the initial upfront purchase and installation investment has been made and maintenance appears nearly free due to centralized support staff paid by institutional overhead, the balance can tip strongly in favor of a local cluster.

	 Traditional Cluster
<p><u>Advantages</u></p> <ul style="list-style-type: none"> ✓ No initial or ongoing support costs ✓ No competition for computing resources ✓ Rapidly and easily scaled for small to large data sets ✓ Resilient, secure distributed system 	<p><u>Advantages</u></p> <ul style="list-style-type: none"> ✓ Exceptional I/O, CPU, network performance ✓ No recurring usage costs ✓ Instances immediately available ✓ Host MPI Applications ✓ Full control of policy and usage
<p><u>Disadvantages</u></p> <ul style="list-style-type: none"> – Data transfer times – I/O, CPU limitations of EC2 instances – Every usage has a cost – Costs start out low, accumulate rapidly – Budget considerations, payment options 	<p><u>Disadvantages</u></p> <ul style="list-style-type: none"> – High startup costs and time – Scaling requires purchase of additional servers and time – Single point of failure – Requires local IT maintenance – OS/Hardware/Grid Software lock-in

Supplementary Figure 10. Comparison of the advantages and disadvantages of AWS cloud computing vs. a traditional in-house compute cluster. The main consideration in deciding which option is better for a given application may be how adept the local users group or institution is at mitigating the disadvantages of a traditional cluster.

SS5: EC2 Spot Instances

For greater economy in using the TPP in cloud services, the ability to minimize the cost of usage by processing during periods of low cloud usage would be a big advantage. Amazon Web Services provides such a cost-saving feature called spot pricing, an attempt to turn computing capacity into another market-driven commodity. A user can gain guaranteed access to an EC2 node at a fixed price as shown in Supplementary Table 1. However, if guaranteed access is not an important factor, then a user can bid an hourly rate that they are willing to pay. When the spot price of the desired nodes is below the bid price, then requested nodes are started. When the spot price is higher than the bid price, then the requested nodes are deferred until the price drops. In this scenario, users pay the spot price at the time the node is started, not the bid price. Therefore, one can bid higher than the spot price, but still pay the spot price for processing. Further, if the spot price goes above the bid price, it is possible (but not guaranteed) that an already-started node will be terminated to satisfy some other usage demand. Hence, while spot pricing can lead to significant savings, there are risks that jobs can be hindered by terminated nodes, and the job may not be completed as quickly as possible. Having just described this system, we note that with AWS, it is not completely transparent and understood, and there are observed anomalies in its behavior. For example, there is a tendency for large jobs to force up the spot price to its bid price, and it has been observed that a low bid price is likely to get the job done nearly as fast, and much more cheaply, than a higher bid price.

There still remains some effort to understand the vagaries of this market. However, the benefit of the *amztp* toolset is that it is designed to be tolerant of the faults in the system, with automatic retries for file transfer failures, nodes that are terminated due to rising demand, and other complications when using commodity computing capacity. The *amztp* system will complete a job at the spot pricing below the bid price, although it may take longer than if guaranteed node pricing was used. In our experience thus far, the cost savings can be significant (a factor of 5 or more), and the additional delay incurred is small. We note that when the individual search jobs are much shorter than an hour, there can be a cost benefit for nodes to be terminated prematurely since they can still complete some work, but there is no charge for the hour during which a node is terminated because the node is needed by a higher bidder. Consider an example where every started node is always terminated by Amazon before completing its first hour: all work could conceivably be done in the segments before termination and cumulatively complete all work at no cost, albeit with considerable time. There is no benefit when the individual searches take much longer than an hour. In the end, if it is most important to minimize cost, and analysis time is by far a secondary concern, it appears most efficient to bid just at the current spot price.

SS6: Application of AMZTPP on a Large MS/MS Dataset

To evaluate the cost effectiveness and performance of AWS for peptide identification in a real-world example, the *amztp* client program was employed on a relatively large dataset consisting of 1110 mzML files containing MS/MS data collected from *Canis familiaris* samples. This dataset was organized in 41 folders and was collected from an LTQ Orbitrap Classic (719 files), an LTQ-Velos Orbitrap (7 files), an LTQ (288 files) and an LCQ Deca (96 files) for a total of 4.3 million MS/MS spectra. The data were searched using X!Tandem, Comet, MyriMatch, and OMSSA for a total of 4440 searches. The searches were performed EC2 spot instances of type c1.xlarge with a maximum spot price (--ec2-spot) of \$0.112 (5 times cheaper than the \$0.58 price of a reserved instance). At the time the average market price was \$0.112 as reported by the AWS console. The maximum number of EC2 instances to start (--ec2-max) was set to 200 and the maximum number of parallel file upload/download processes (-P) was set to 10. Results of all searches were then run through TPP's PeptideProphet, iProphet, and ProteinProphet tools for further statistical analysis of the peptide and protein identifications. To calculate the total cost, a simple script was written to query the Amazon's account usage report interface and generate reports for SQS, S3, and EC2 before and after the execution of the searches and the difference between the records used in the cost calculations.

Supplementary Table 3 displays the cost breakdown for running the 4440 canine search dataset for all EC2, S3, and SQS usage including data transfer in and out. The total cost for searching all data with four different search algorithms was \$88.12 and the searches were completed in 9.2 hours using 654 machine-hours, with 77% of the cost being represented in EC2 spot instance time.

Elastic Compute Cloud (EC2) Costs

<u>Operation</u>	<u>Spot Price</u>	<u>Hours/Usage</u>	<u>Cost</u>
c1.xlarge (S0001)	\$0.112	555	\$ 62.16
c1.xlarge (S0002)	\$0.112	78	\$ 8.74
c1.xlarge (S0004)	\$0.112	21	\$ 2.35
PublicIP-In	0	25,872	\$ -
PublicIP-Out	\$ 0.12/GB	11,228	\$ 0.00
InterZone-In	0	442,308	\$ -
InterZone-Out	\$ 0.02/GB	1,920	\$ 0.00
<i>First 1 GB / month data transfer out is free</i>			EC2 Total: \$ 73.25

Simple Storage Service (S3) Costs

<u>Operation</u>	<u>Price</u>	<u>Usage</u>	<u>Cost</u>
PUT, COPY, POST, or LIST	\$ 0.005/1,000	12,776	\$ 0.06
GET, other requests	\$ 0.004/10,000	42,583	\$ 0.02
Data Transfer In (GB)	\$ -	254.2	\$ -
Data Transfer Out (GB)	\$ 0.12/GB	113.6	\$ 13.63
Data Storage (GB)	\$ 0.095/GB mth	367.8	\$ 1.13
<i>First 1 GB / month data transfer out is free</i>			S3 Total: \$ 14.84
<i>Data storage cost based on 24 hours</i>			

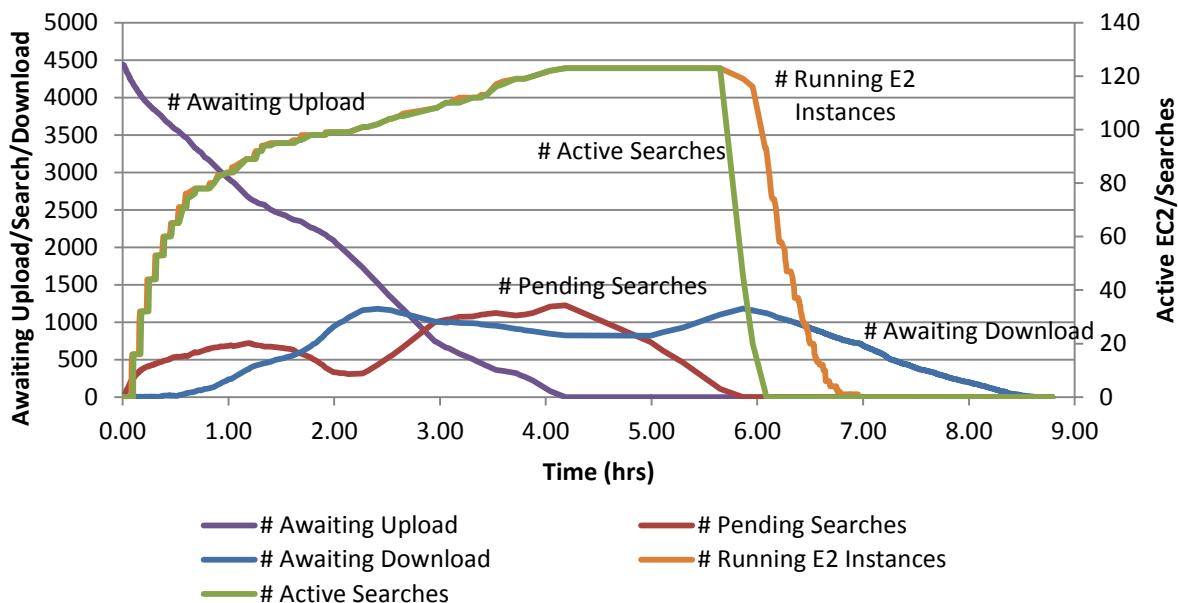
Simple Queue Service (SQS) Costs

<u>Operation</u>	<u>Price</u>	<u>Usage</u>	<u>Cost</u>
Requests	\$ 0.05/100,000	56,030	\$ 0.03
Data Transfer In	\$ -	0	\$ -
Data Transfer Out	\$ 0.12/GB	0.04	\$ 0.00
<i>First 1M requests are free</i>			SQS Total: \$ 0.03
<i>First 1GB out is free</i>			
			Grand Total: \$ 88.12

Supplementary Table 3. Cost breakdown for searching the Canine dataset of 1110 MS runs through four search engines on the AWS. All prices are given in USD.

Supplementary Figure 11 depicts several metrics of the four-engine searches of 1110 MS runs as a function of time. The purple line indicates the number of searches awaiting upload. All 4440 searches were uploaded by 4.2 hours into the job, indicating an approximate upload speed of 13.7 seconds per MS search; a total of 1,244 files were uploaded (duplicates are excluded and mzML files are compressed) equaling 9,108 MB and this translates to ~2.49 MB/s upload. The orange line indicates the number of running nodes on EC2 and the green line indicates the approximate number of active searches. By ~6 hours into the analysis all searches have been completed. By 6.96 hours all nodes have shut down (just under 1 hour after the last search completed in order to avoid an extra hour of billing). Downloading output files continues for another 1.7 hours. The red line shows the approximate number of searches in the queue. At the beginning of the time

sequence this number grows until enough EC2 instances are provisioned to manage the influx of searches, as shown by the plateau and slight decline. Later idle nodes are indicated by the difference in active searches vs EC2 searches and subsequently these nodes are terminated as they near the next hour billing mark. A second increase in searches in the queue occurs at the 3 hour mark, presumably caused by a sharp increase in the job submission rates due to the majority of files to search having already been uploaded. The final blue line indicates the number of finished search results left to download. This value mostly increases during the first 2 hours, modulated by the upload speed, and then decreases slowly for 1.7 hours after all data are searched.



Supplementary Figure 11. EC2 provisioning results for the search of 4440 jobs of the Canine dataset. The purple, red, and blue lines indicate the approximate number of searches waiting to upload, the pending searches, and number of results awaiting download, respectively (left axis labels). The green and orange lines indicate the number of spot instances that were running and the approximate number of searches active, respectively (right axis labels).

SS7: Effect of upload/download speeds on total processing time

Since proteomics data searching is both a data intensive process as well as a CPU-intensive process, a practical limit in the number of simultaneous searches that can be performed is proportional to the ratio between the upload speed and the processing speed. For example, if each run takes 10 minutes to upload and 10 minutes to search, then it will be difficult to keep more than one EC2 instance busy. Often parallel uploads can boost the overall throughput over serial uploads, but this is typically limited to a factor of a few in improvement. Clearly more complex searches with more modifications and larger search spaces will benefit more greatly from cloud computing. For example, if uploads take 5 minutes per file and the searching takes an hour per file, a dozen nodes can easily

be utilized. Download speed can also have an impact, but search results tend to be much smaller than the input spectra, and therefore constitute a much smaller factor.

Note that if one is using the TWA hosted solution, and the files are already in S3 storage, then local upload speed is not an important factor and large amounts of parallelization can occur. There still remains a finite transfer speed between S3 nodes and compute nodes, so one still cannot scale to a huge number of nodes. Clearly there are also some speed improvements to be gained in reducing the size of the spectral information that must be transmitted to S3. In the amztp search workflow, mzML files are uploaded compressed for improved efficiency. Additional efficiencies such as stripping out MS1 spectra in the data upload to S3 can help for datasets where quantitation software will not be executed on EC2.

Supplementary Material References

1. Ou Z, Zhuang H, Nurminen JK and Ylä-Jääski A (2012) Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2. 4th USENIX Workshop on Hot Topics in Cloud Computing
<https://www.usenix.org/conference/hotcloud12/exploiting-hardware-heterogeneity-within-same-instance-type-amazon-ec2>.