TheCloudExchange

A Futures Market for Spot Instances

Prepared by:
Alan Verga akv2001@columbia.edu
Evgeny Fedotov ef2363@columbia.edu
Suhan Canaran sc3055@columbia.edu

Introduction and motivation

Spot Instances are a new type of service offered by Amazon EC2, in addition to On-Demand and Reserved Instances. Spot Prices change periodically based on supply and demand, and customers who bid in excess gain access to the available Spot Instances. Generally, Spot Prices are lower than On-Demand prices which allows customers to pay less. Nevertheless, the prices can fluctuate significantly depending on the day and time (see the picture below).

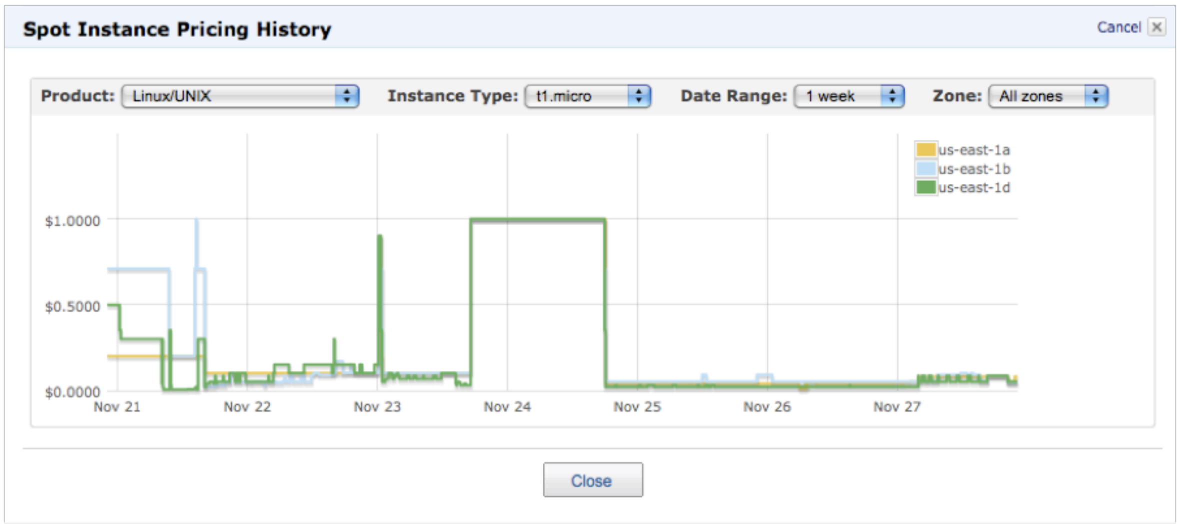


Figure 1. Spot instance price fluctuation.

TheCloudExchange is an automated exchange platform for trading Spot Instance futures contracts. A Spot Instance Futures Contract is a contract between two parties to exchange ownership over an EC2 Spot Instance at specified date in future. Possible users of TheCloudExchange can be companies that would like to fix their computational costs in future (hedge against unpredictable price moves for Spot Instances) and speculators betting on a particular price trend.

TheCloudExchange is an online application run on the Google App Engine platform. Users can view bid/ask prices for all available contracts and place their bids and offers for specific contracts in future. Upon deal execution, TheCloudExchange asks parties to exchange cash flows and send confirmation emails. At delivery time, TheCloudExchange launches a Spot Instance on behalf of a buyer at a currently available Spot Price. If the price is higher than the negotiated price, TheCloudExchange debits the sellers account for the deficient amount.

Futures Contract specification

Regular exchange contracts will specify the following:

- Instance size (micro, small, large)
- Instance type (Linux, Windows, Suse)
- Region (US East, EU, Asia)
- Zone (us-east-1a, us-east-1b)
- Delivery date and time (eg. December 30, 2011 2pm)
- Duration (1 hour intervals)

Case study
Consumers have a batch processing job to run in the future (1 or 2 weeks from now). The job will take a number of hours or days to run, but is flexible in when it can be started and completed, so you would like to see if you can complete it for less cost than with On-Demand Instances. You have a limited budget for this task.
Producers have excess, idle computational resources. Placing offers on TheCloudExchange can monetize unused capacity.

Speculators can provide added liquidity and risk transferrence through market participation.

Solution: All participants can go to TheCloudExchange and purchase the best combination of futures contracts available to meet their requirements.

System architecture

1. Overall Architecture
TheCloudExchange is online application written in Java and running on the Google App Engine platform.

We utilize the MVC (Model-View-Controller) design approach to build the application, as follows:

- View: JSP pages with jQuery scripts were used on the front-end
- Controller: Servlets were used as a controller that negotiated with the Datastore and prepared the data for displaying on the client
- Model: the data was stored in the Datastore entities, retrieved and formatted by Servlets

The application is deployed on http://thecloudexchange.appspot.com.

2. App Engine Features
As the part of the project the following core App Engine features were implemented:

- Authentication through Google accounts API
- Datastore & Memcache for persisting objects
- Blobstore for persisting the AWS credentials file
- Channels for real-time messaging between the clients
- E-Mails for notifying the customers about executed deals and delivered instances
- Task Queues for simultaneous execution of long-running jobs (launching spot instances)
- Cron jobs that run every hour to see whether any instances are scheduled to be delivered and subsequently launching the spot instances passing the launching configuration and parameter to the workers in task queue

3. Package & Classes Structure
Our project was split in the following packages (all under edu.columbia.e6998.cloudexchange* name space):

- *.aws: Classes/servletes used for automatically launching AWS Spot Instances
- *.aws.spotprices: Specific classes/servlets used to retrieve Spot Instance prices from Amazon
- *.channel: Channel messaging classes/servlets
- *.client: Classes/servlets used to display information on the client side in main.jsp and account.jsp pages.
- *datastore: Clases/servlets used for storing & retrieving entities from Datastore and Memcache
- *mailer: Contains MailManager class used for sending confirmation e-mails to the traders
- *toolkit: Auxiliary classes for Datastore access

4. Datastore Entities
The below diagram represents the relationships between Datastore entities used for

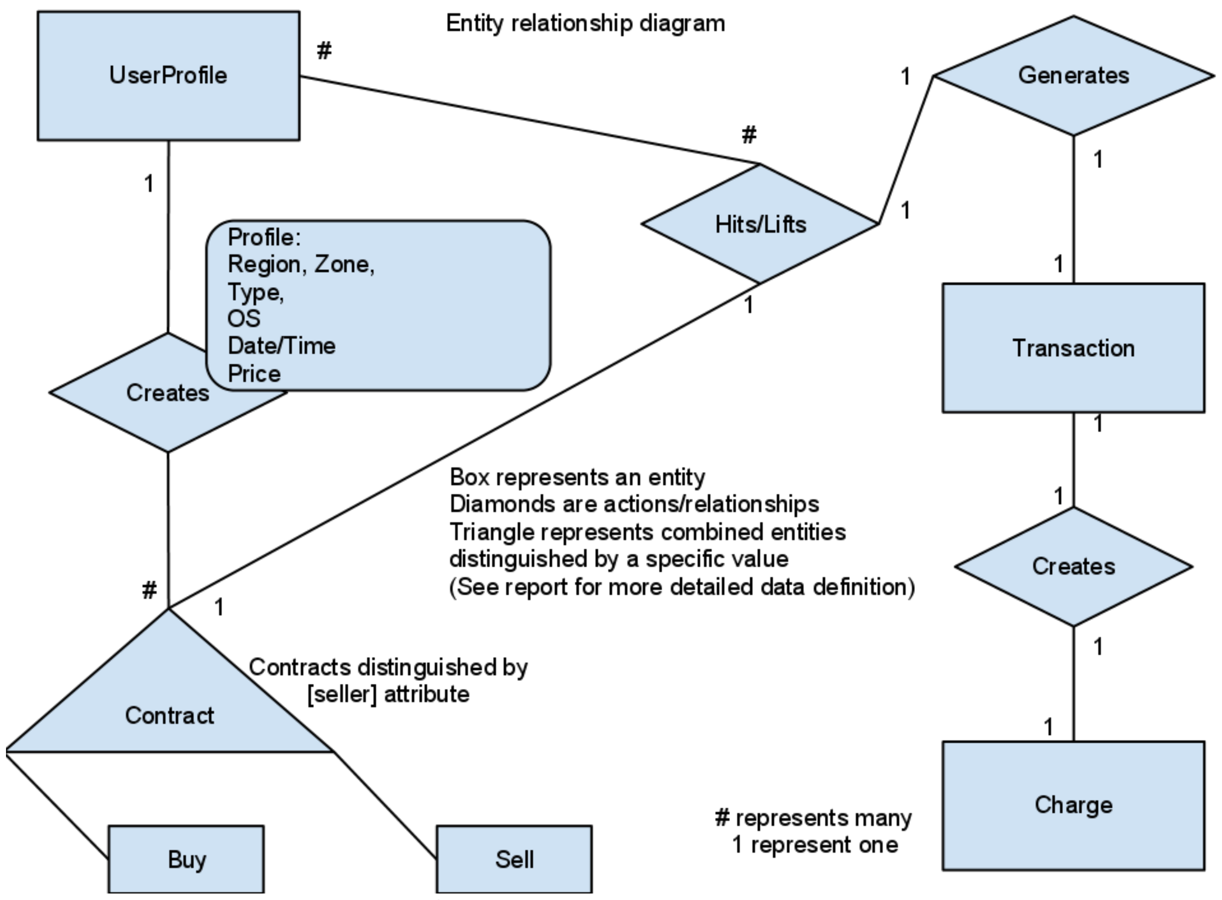


Figure 2. Datastore Entity Relations Diagram

UserProfile: represents a Google user ID, default Amazon Key Pair and Security Group, and a reference to a Blobstore object that contains user's AWS credentials

Contract: Buy/Sell orders created by users. Uses profile as key, also the userID, and time
A user can create may contracts.
Once a user buys an order (hit/lift contract) it generates a transaction.
Transaction: contract details plus interested party information is stored.
Once the scheduled time for a transaction happens it creates a charge

Charge contains the money owed to other interested party that is resulting from a successful launch of an instance at the agreed hour.

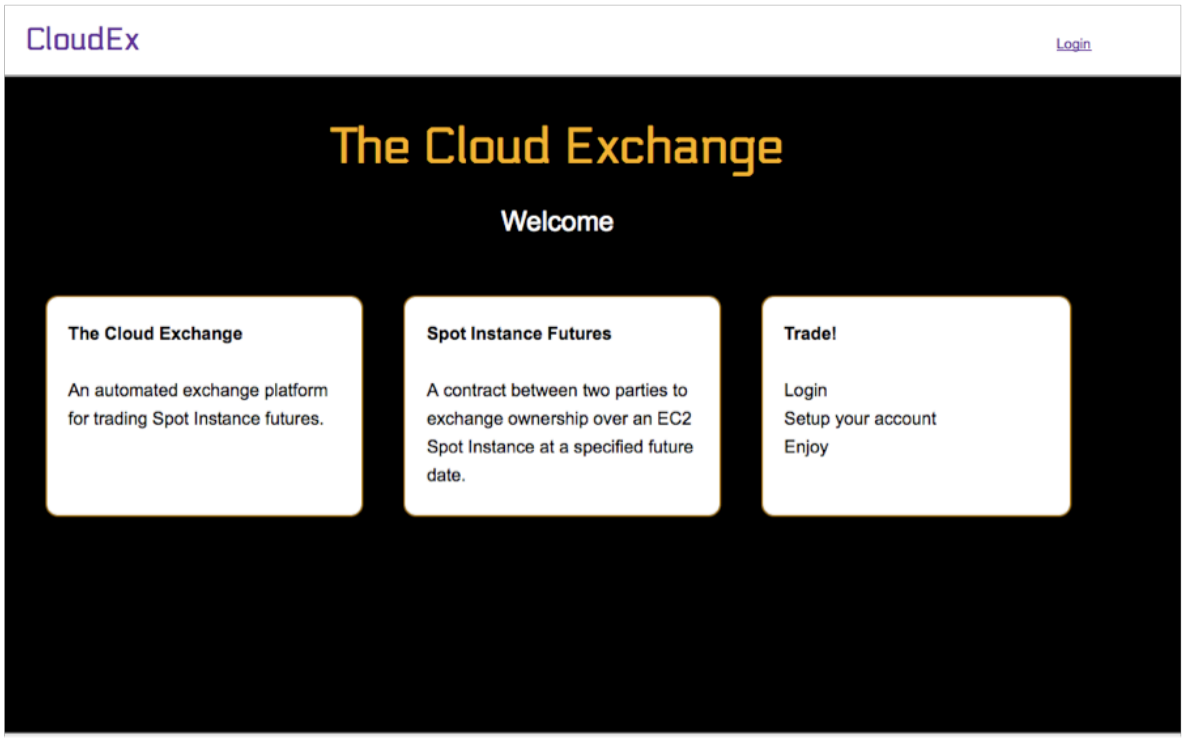
Datastore Challenges & Memcache:

To get around the slow Datastore response we implemented some basic methods to deal with data in Memcache directly. Data that should be displayed to the user in the front end would always get stored in the Memcache - that is the best bids and offers available at any given time.

When a new bid/offer gets created or a transaction occurs, we developed functions to determine if these newly created records should change the data in Memcache. If so, Memcache gets updated first, a message to all clients gets send, and write request to Datastore gets initiated. This way we could show close to real time data processing even in the cloud context.

Implementation

Landing Page
Landing page has basic description of the application features and a login link. Simple but necessary.



The Cloud Exchange

An automated exchange platform for trading Spot Instance futures.

Spot Instance Futures

A contract between two parties to exchange ownership over an EC2 Spot Instance at a specified future date.

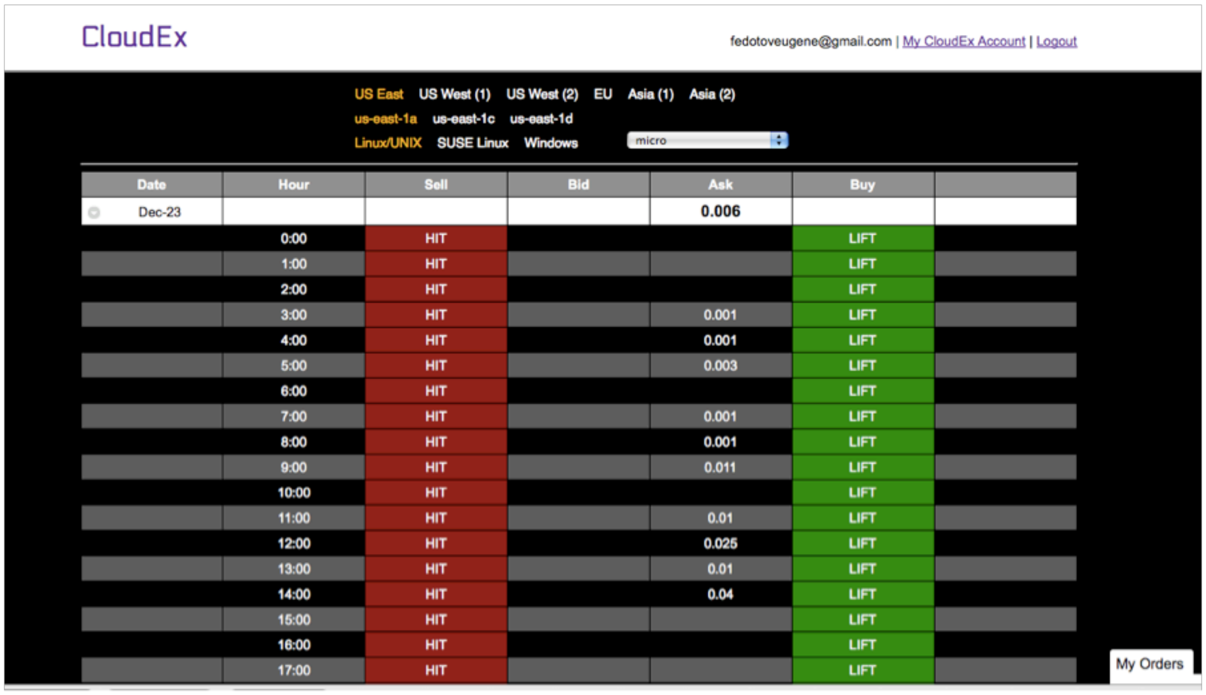
Trade!

Login
Setup your account
Enjoy

Figure 3. Landing page

Trading Page
Trading page is the place where you can trade futures contracts by placing bids (order to buy a contract for a specific price) and offers (order to sell a contract for a specific price). Generally, offer prices (also referred to as “ask prices”) are higher than the bid prices.

Traders can choose the contracts they would like to trade by selecting Regions, Zones, Operating System and Instance Size in the upper menu of the page. The order table below will be automatically filtered to display the contracts meeting the aforementioned criteria. The table displays bids/offers ordered by date and time. The date row (highlighted in white) shows the current price for a given Spot Instance. Users can expand date rows to view an hour-per-hour picture.


CloudEx

fedotoveugene@gmail.com | My CloudEx Account | Logout

US East US West (1) US West (2) EU Asia (1) Asia (2)

us-east-1a us-east-1c us-east-1d

Linux/UNIX SUSE Linux Windows

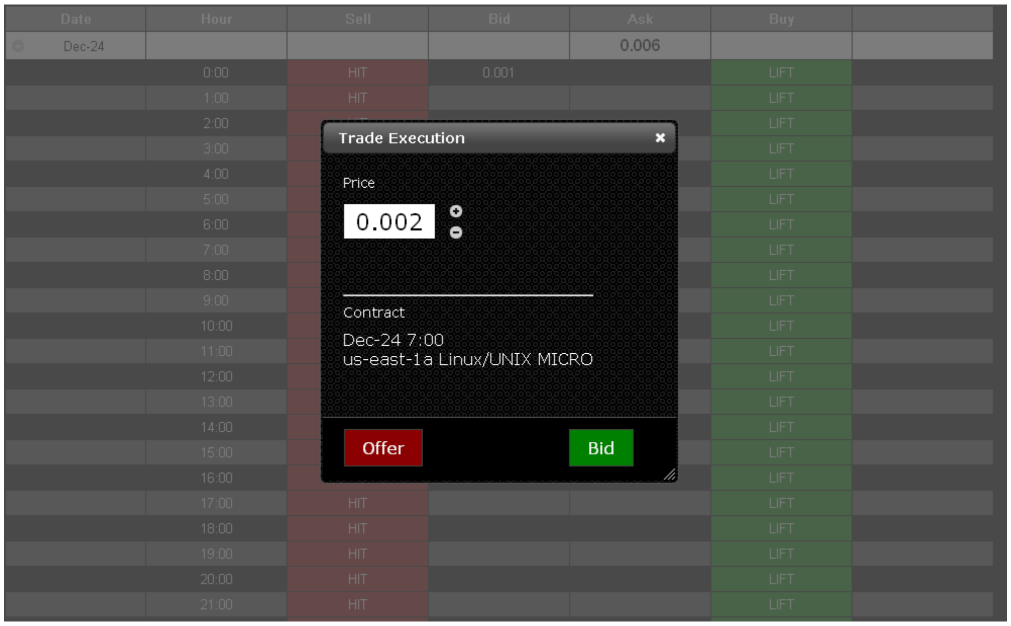
micro

Date	Hour	Sell	Bid	Ask	Buy	
Dec-23				0.006		
	0:00	HIT			LIFT	
	1:00	HIT			LIFT	
	2:00	HIT			LIFT	
	3:00	HIT		0.001	LIFT	
	4:00	HIT		0.001	LIFT	
	5:00	HIT		0.003	LIFT	
	6:00	HIT			LIFT	
	7:00	HIT		0.001	LIFT	
	8:00	HIT		0.001	LIFT	
	9:00	HIT		0.011	LIFT	
	10:00	HIT			LIFT	
	11:00	HIT		0.01	LIFT	
	12:00	HIT		0.025	LIFT	
	13:00	HIT		0.01	LIFT	
	14:00	HIT		0.04	LIFT	
	15:00	HIT			LIFT	
	16:00	HIT			LIFT	
	17:00	HIT			LIFT	

My Orders

Figure 4. Trade page

Traders are able to enter bids and offers by clicking on the desired hour row. A window describing the contract lets the user input their bid and offer. Quantities default to one contract.


Date	Hour	Sell	Bid	Ask	Buy	
Dec-24				0.006		
	0:00	HIT	0.001		LIFT	
	1:00	HIT			LIFT	
	2:00				LIFT	
	3:00				LIFT	
	4:00				LIFT	
	5:00				LIFT	
	6:00				LIFT	
	7:00				LIFT	
	8:00				LIFT	
	9:00				LIFT	
	10:00				LIFT	
	11:00				LIFT	
	12:00				LIFT	
	13:00				LIFT	
	14:00				LIFT	
	15:00				LIFT	
	16:00				LIFT	
	17:00	HIT			LIFT	
	18:00	HIT			LIFT	
	19:00	HIT			LIFT	
	20:00	HIT			LIFT	
	21:00	HIT			LIFT	

Trade Execution

Price

0.002

Contract

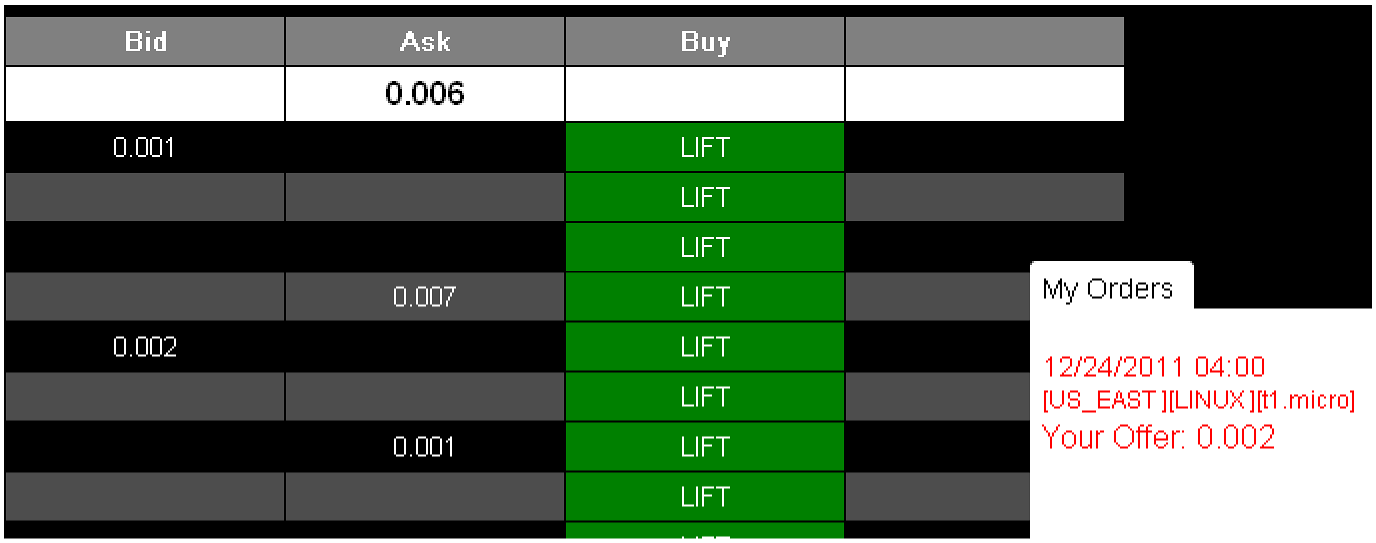
Dec-24 7:00
us-east-1a Linux/UNIX MICRO

Offer Bid

Figure 5: Entering Bid / Offer

Traders can enter into contracts by matching existing bid to offers, thereby creating long and short positions on a specific futures contract. Following common trading terminology, matching an existing bid is called "hit" and is marked by red color button on the trading screen whereas matching an existing offer is called "lift" which is highlighted in green.

An order book displays the current users unfilled bids or offers. It is minimized to the bottom right hand corner, and pops up when clicked. Orders are cancelled by clicking on the order in the “My Orders” tab.



Bid	Ask	Buy	
	0.006		
0.001		LIFT	
		LIFT	
		LIFT	
	0.007	LIFT	
0.002		LIFT	
		LIFT	
	0.001	LIFT	
		LIFT	

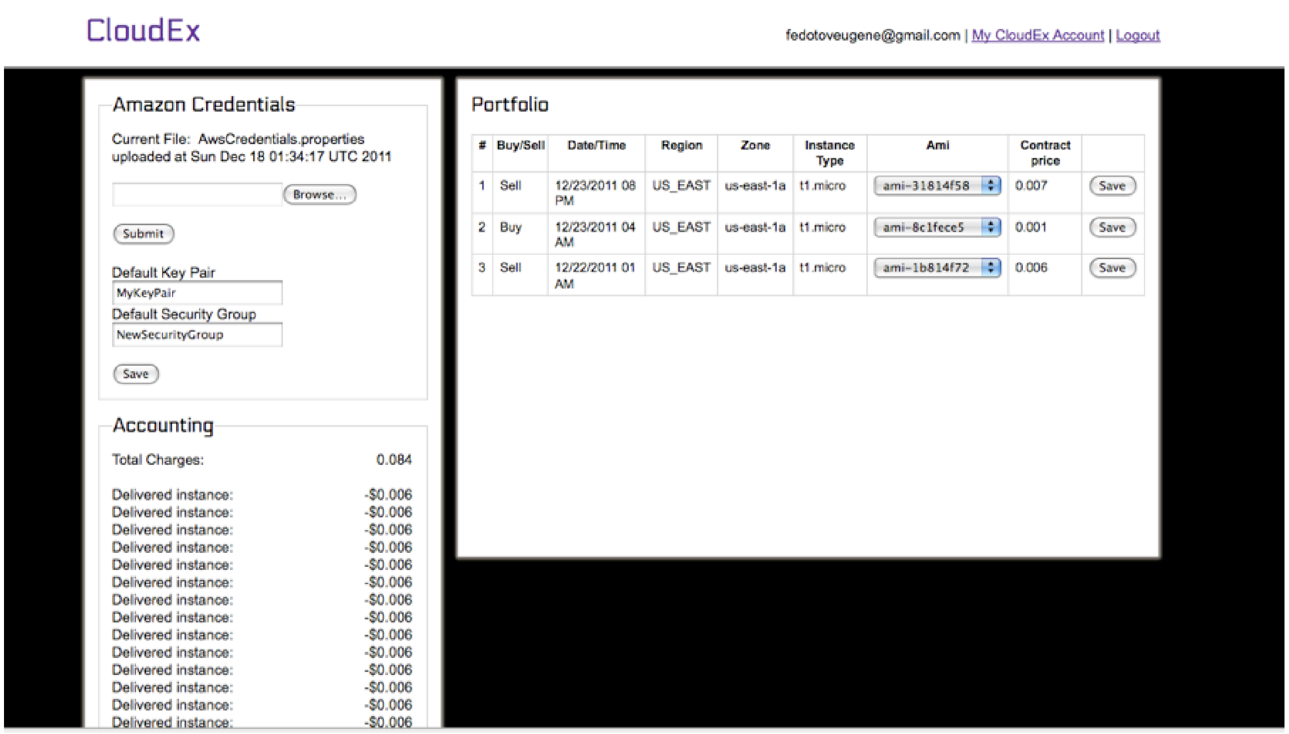
My Orders

12/24/2011 04:00
[US_EAST][LINUX][t1.micro]
Your Offer: 0.002

Figure 6: Cancelling an Order

Account Page
Account page is a page for viewing and managing user accounts. It consist of three separate parts:

- Amazon Credentials: in this section users upload and input the data needed to launch Spot Instances automatically on their behalf. The most important step is uploading the AWS credentials file which is stored using BlobService. **Note:** Users are urged to upload their AWS credentials file prior to executing any deals.
- Portfolio: shows current open short/long (sell/buy) positions. Users can choose the ami to be launched when a Spot Instance is delivered from a dropdown box for each position.
- Accounting: displays a list of charges for the account. Sellers are charged when a Spot Instance is delivered to buyers. Buyers are charged when buying a specific futures contract.



CloudEx

fedotoveugene@gmail.com | My CloudEx Account | Logout

Amazon Credentials

Current File: AwsCredentials.properties
uploaded at Sun Dec 18 01:34:17 UTC 2011

Default Key Pair: MyKeyPair
Default Security Group: NewSecurityGroup

Accounting

Total Charges:	0.084
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006
Delivered instance:	-$0.006

Portfolio

#	Buy/Sell	Date/Time	Region	Zone	Instance Type	Ami	Contract price	
1	Sell	12/23/2011 08 PM	US_EAST	us-east-1a	t1.micro	ami-31814f58	0.007	Save
2	Buy	12/23/2011 04 AM	US_EAST	us-east-1a	t1.micro	ami-8c1fece5	0.001	Save
3	Sell	12/22/2011 01 AM	US_EAST	us-east-1a	t1.micro	ami-1b814f72	0.006	Save

Figure 7. Account page

Conclusions

To summarize, we decided to show the efficiency of using Spot Instance futures on a specific example of computational costs connected with primitive sorting operations.

	Per hour, $		**3-year cost, $**	
On-Demand Prices	**Linux**	**Windows**	**Linux**	**Windows**
micro	0.02	0.03	525.60	788.40
small	0.085	0.12	2,233.80	3,153.60
large	0.34	0.48	8,935.20	12,614.40
xlarge	0.68	0.96	17,870.40	25,228.80
Spot Prices				
micro	0.006	0.012	157.68	315.36
small	0.027	0.045	709.56	1,182.60
large	0.108	0.18	2,838.24	4,730.40
xlarge	0.216	0.36	5,676.48	9,460.80

Table 1. Comparison of On-Demand vs. Spot Instance costs

Working backwards from transaction processing metrics and how they calculate a penny's worth system time[1], we came up with the numbers above showing us the approximate system value of the systems that Amazon instances that we lease in On-Demand fashion.

The On-Demand 3-year prices and the machine specifications show us a significant difference (up to 65% decrease in costs) between On-Demand and Spot Instance computational costs.

We do agree that on demand services are not necessarily meant for long term uses since the costs are not fully justified. However, using a spot instance on demand shows real promise of bringing down the costs, and buying your money's worth of computation on demand. This would definitely be the next steps of CloudEx since in our project we aimed to promote usage of spot instances in a futures market manner. We added an incentive to buy and trade spot instance futures allowing users to protect themselves against price fluctuations.

References

[1] A measure of transaction processing power. Datamation, 31(7):112-118, 1985. Also in: Stonebraker, M.J., ed. Readings in Database Systems. San Mateo, CA: Morgan Kaufmann, 1989.