Uma análise do impacto das plataformas *pay-as-you-go* de computação em nuvem na construção e precificação de software

Fernando Pires Barbosa[1] Andrea S. Charão[2]

Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

Resumo

Cloud Computing é um novo paradigma que está mudando a forma com que consumimos recursos de TI. Há vários tipos de serviços oferecidos no modelo *Cloud* e o nosso foco neste trabalho está relacionado a Plataformas como Serviço (*PaaS*). Realizamos uma análise do impacto que as plataformas *pay-as-you-go* irão gerar no desenvolvimento de software e identificamos mudanças significativas relacionadas à forma como a otimização e precificação de software deverão ser encaradas. As mudanças provocadas pelo modelo *pay-as-you-go* na área de desenvolvimento de software tem sido pouco exploradas e esperamos que este trabalho contribua para que novos estudos sejam realizados sobre este tema.

Palavras-chave: Computação em nuvem, precificação de software, Plataformas *Cloud* , otimização de código

Abstract

Cloud Computing is changing the way we use IT resources. There are diferent kinds of service being offered through Cloud model and we focused our work at Platform as a Service (PaaS). We analyzed how the pay-as-you-go platforms will impact the software development in the next years. Based on these analysis, we identified significant changes at code improvement activities and software pricing policies. The impact generated by the pay-as-you-go model in software development has been less achieved and we hope this article can estimulate this area.

Keywords: *Cloud* Computing, software pricing, *Cloud* platform, code improvement

[1] Email: fernando.pires.barbosa@gmail.com
[2] Email: andrea@inf.ufsm.br

1 Introdução

Nos últimos anos o tema *Cloud Computing* vem sendo objeto de estudo na comunidade científica por abordar um novo paradigma que deverá mudar a forma como consumimos recursos de TI[4]. Esta mudança já chegou ao mercado (um exemplo é o *Amazon S3*[1]) e atualmente é possível contratar recursos de TI através da internet na casa de centavos de dólar por hora de uso sem preocupar-se com a localização ou disponibilidade. Este modelo de cobrança também é chamado de *pay-as-you-go*, uma vez que o preço a ser pago é função do volume de recurso usados, semelhante ao consumo de energia elétrica.

Há vários tipos de serviço que podem ser oferecidos no modelo *Cloud Computing* e várias classificações tem sido feitas para agrupá-los conforme a semelhança entre as suas características, com destaque para itens como *IaaS* (Infraestrutura como Serviço), *PaaS* (Plataforma como Serviço) e *SaaS* (Software como Serviço). Neste trabalho abordamos os impactos que o fornecimento de serviços do tipo *PaaS* podem gerar no desenvolvimento de software levando em consideração o mecanismo de cobrança *pay-as-you-go* .

Na seção 2 fazemos uma revisão dos conceitos, tipos de serviço e desafios relacionados a *Cloud Computing*. Na seção 3 fazemos um levantamento das plataformas *Cloud* existentes, analisando a forma como os desenvolvedores de software podem utilizá-las e como elas estão tratando as principais questões relacionadas ao modelo *pay-as-you-go*, enquanto na seção 4 descrevemos algumas experiências que realizamos para experimentar a utilização deste modelo no desenvolvimento de software. Na seção 5.1 fazemos uma análise do impacto que o modelo pode gerar no desenvolvimento de software e, com base nesta análise, iniciamos o estudo de um modelo 5.2 que leve em consideração os impactos que identificamos. A seção 6 conclui o trabalho, resumindo os resultados obtidos e indicando um rumo para novas pesquisas nesta área.

2 Cloud Computing

A divulgação cada vez maior do termo *Cloud Computing* vem chamando a atenção dos profissionais de TI com a promessa de ser um novo paradigma[4] que irá mudar a forma como os recursos de TI são comercializados, possibilitando que a distribuição e o consumo destes recursos sejam realizados nos mesmos moldes em que usamos serviços de água e energia elétrica atualmente.

Apesar de nova, a idéia de possibilitar o acesso a recursos de TI nestes moldes não é inédita, uma vez que este mesmo conceito está na origem da criação dos ambientes de *Grid Computing* [5], que teve seu desenvolvimento durante a década de noventa. Entretanto, a promessa original não foi cumprida e os ambientes em *Grid* acabaram evoluindo segundo as necessidades das instituições científicas, que precisam de altas capacidades de desempenho

computacional para realizar determinados tipos de experimentos. Estas instituições encontraram no modelo de *Grid* uma forma de compartilhar entre si os recursos que cada uma possui a sua disposição.

Economia de escala e relação com *Grid Computing*

Cloud Computing utiliza alguns dos conceitos e também algumas das tecnologias originadas durante o desenvolvimento dos ambientes *Grid*, mas apresenta também alugmas diferenças [5]. O contexto tecnológico disponível durante o período de desenvolvimento dos dois modelos pode ter dado origem a uma diferença que, em nossa opinião pode estar impulsionando o modelo *Cloud* na direção das expectativas que surgiram com o modelo *Grid*. Esta diferença está relacionada ao modelo de negócio. Enquanto *Grid* está baseado no compartilhamento de recursos para benefício mútuo, o modelo de negócio de *Cloud* está baseado na venda e utilização sob demanda destes mesmos recursos. A idéia desse novo modelo de negócio é fornecer recursos infinitos que podem ser utilizados pagando-se uma taxa conforme o volume de utilização, assim como fazemos hoje com água e energia elétrica (o termo *Grid Computing*, aliás, é uma analogia com as grades de fornecimento de energia[4]).

Este modelo está baseado em economia de escala. O fornecedor da *Cloud* investe na construção de uma infra-estrutura computacional que proporcione a ilusão de um volume infinito de recursos e os comercializa para um número de clientes alto o suficiente para tornar a operação rentável. Por ser um termo recente, ainda não há um consenso sobre o significado de *Cloud Computing*. Em sua avaliação sobre *grid* e *cloud*, Ian Foster[5] introduz a seguinte definição:

"Cloud Computing é um paradigma de computação em larga escala que possui foco em proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da Internet."

Atualmente existem vários serviços disponíveis e sendo comercializados no modelo *Cloud*, indicando que as idéias propostas já estão em prática, ainda que não tenham alcançado todo o potencial de crescimento disponível.

Tipos de serviços oferecidos em *Cloud Computing*

Com a diversidade de serviços que passaram a ser oferecidos pelos fornecedores, várias classificações tem sido propostas para esclarecer as diferenças entre eles. Uma delas está em um trabalho da HP [7], resumido na figura 1.

Um desenvolvedor de software que queira construir e distribuir aplicações no modelo *Cloud* pode fazer isso tomando como base qualquer um dos diferentes tipos de serviço existentes. A figura 2 mostra como uma aplicação pode ser desenvolvida e distribuída considerando os diferentes tipos de serviço como camadas de abstração. É possível construir uma aplicação e hospedá-la em um fornecedor *IaaS*, responsabilizando-se por manter as devidas configurações dos recursos de hardware e software relacionados. Outra opção seria um fornece-

Tipo	Resumo	Exemplo
Infrastructure as a Service ***(IaaS)***	Disponibilização de recursos de hardware, como espaço em disco e capacidade de processamento	*Amazon S3*
Data as a Service ***(DaaS)***	Um tipo especializado de armazenamento que envolve serviços de Banco de Dados	*Amazon SimpleDB*, *Google Big Table*
Platform as a Service ***(PaaS)***	Provê serviços e APIs para facilitar o desenvolvimento e distribuição de aplicações	*Google AppEngine, force.com, Microsoft Azure AppFabric*
Software as a Service ***(SaaS)***	Provê aplicações inteiras acessadas diretamente pela Internet, sem necessidade de downloads ou aquisição de licenças.	*GMail*, *Salesforce.com*, *Google Docs*

Fig. 1. Classificação dos tipos de serviço oferecidos em *Cloud* Computing

dor *DaaS*, onde podem ser utilizadas as facilidades de serviços de banco de dados. Entretanto, a opção mais natural seria a utilização de um fornecedor *PaaS*, uma vez que além da infra-estrutura eles em geral oferecem serviços que facilitam a construção, distribuição e acompanhamento do software.

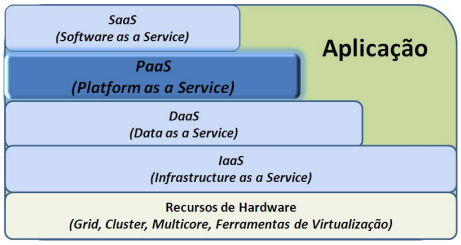

Fig. 2. Camadas de abstração de serviços Cloud Computing, com destaque para *PaaS*

Desafios relacionados a *Cloud Computing*

Em 2009 a Universidade de Berkeley publicou um trabalho [11] em que listava os principais desafios em *Cloud Computing*, citando itens como segurança, disponibilidade, escalabilidade e licenciamento, resumidos na figura 3.

Desafio	Resumo
Disponibilidade dos Serviços	Problemas do *data center*, problemas da aplicação, problemas com a Internet.
Data lock-in	Ficar preso a um Fornecedor (dados e/ou aplicações)
Segurança e Confidencialidade	Certeza de que os dados não serão acessados por terceiros. Questão mais cultural do que técnica.
Gargalos para transmissão de dados	A tecnologia para transmissão de dados tem evoluído mais lentamente que a tecnologia de armazenamento e processamento.
Escalabilidade e desempenho	"Virtualização" aplicada à capacidade de armazenamento. Forma como as aplicações irão escalar
Debug de erros	Como *debugar* erros visíveis apenas em larga escala?
Licenciamento	Os SLAs no modelo *Cloud* serão diferentes dos tradicionais que existem atualmente

Fig. 3. Resumo dos desafios relacionados a *Cloud* Computing

Tanto a comunidade científica quanto as empresas que fornecem soluções *Cloud* vem evoluindo nesses temas com uma velocidade considerável, o que pode estar sendo impulsionado pela própria disputa de mercado entre grandes empresas como *Amazon*, *Microsoft* e *Google*. Os desafios de licenciamento e escalabilidade possuem relação direta com o modelo *pay-as-you-go* e tem destaque especial na análise que apresentamos na seção 3.

3 Identificação e análise de Plataformas *(PaaS) Cloud*

Como *PaaS* é a opção natural para desenvolvedores que pretendem construir e distribuir software no modelo *Cloud*, fizemos um levantamento e uma breve análise das plataformas disponíveis. Em 2010 a *wikipedia* listava as seguintes plataformas na categoria *Cloud Platforms*[15]: *Amazon Simple Queue Service*, *Amazon SimpleDB*, *Amazon Web Services*, *AppScale*, *MS Azure Services Platform (AppFabric)*, *Caspio*, *Engine Yard*, *Force.com*, *Google App Engine*, *Sun Cloud*, *Heroku*, *Orange Scape*, *Rackspace Cloud*, *Vertebra*, *Visual WebGUI*.

Mesmo categorizadas pela *wikipedia* como *Cloud Platform* e apresentando um comportamento básico que as coloca nesta categoria, elas possuem características específicas que as tornam diferentes entre si, o que identificamos através das informações publicadas em seus respectivos sites. Como nosso objetivo neste trabalho se refere ao impacto destas plataformas na construção de software optamos por identificar, em primeiro lugar, características uniformes quanto ao modo que elas são usadas pelos desenvolvedores (figura 4). A partir de então selecionamos algumas plataformas para uma análise mais detalhada.

Foco da Plataforma	Nome/Fornecedor da Plataforma
Biblioteca de Serviços	Amazon Web Services (infraestrutura)
Criação Rápida de Formulários	Force.com, Caspio, Rollbase
Infra-Estrutura para Ruby on Rails	Heroku, Engine Yard
SDK para Programação e Publicação	Microsoft Azure AppFabric, Google App Engine

Fig. 4. Classificação das plataformas quanto ao modo de utilização *Cloud Computing*

As plataformas *force.com*, *rolbase* e *caspio* são bastante parecidas, permitindo criar formulários de forma rápida dentro de um padrão pré-estabelecido e com modelos de cobrança semelhantes. Para representá-las escolhemos a *force.com*[13], que é uma das pioneiras não apenas em *PaaS*, mas também em *SaaS* com o *salesforce.com*. Não avaliamos as plataformas do tipo "biblioteca de serviços" por considerá-las mais próximas de *IaaS* do que *PaaS*. Avaliamos também a plataforma *Heroku*[8] e as duas plataformas baseadas em SDKs: *AppFabric*[12] e *AppEngine*[6]. O resumo da avaliação está na figura 5.

Plataforma	Foco / Forma de Utilização	Modelo de Cobrança	Escalabilidade	Linguagem
Force.com	Construção de Formulários	Nº de usuários Nº de aplicações e tabelas	Automática	Appex
Heroku	Infra-estrutura p/ Aplicações Ruby on Rails	Infra-estrutura alocada + "add-ons"	Manual	Ruby on Rails
Micosoft Azure AppFabric	SDK para .net + infra-estrutura para hospedagem	Infra-estrutura alocada + "dados sob demanda"	Manual	.net framework
Google App Engine	SDK p/ Java e Python + infra-estrutura para hospedagem	Infra-estrutura e dados sob demanda	Automática	Java, Python

Fig. 5. Quadro comparativo (resumo): *force.com[13], heroku[8], AppFabric[12] e AppEngine[6]*

A análise que fizemos não foi exaustiva, e demos uma atenção especial às características relacionadas ao modelo *pay-as-you-go*. Assim, concentramos

nossa avaliação nos itens relacionados a modelo de cobrança (ou licenciamento) e escalabilidade, tendo como pano de fundo uma analogia com a energia elétrica, que consumimos de forma transparante e pagamos através de uma fatura mensal baseada nos KW consumidos. A figura 6 resume os modelos de cobrança e preços das quatro plataformas analisadas.

Preços *force.com*

Item	Free	Enterprise	Unlimited
Aplicações	1	10	*Free*
Usuários	100	$50/user/mês	$75/user/mês
Objetos/Tabelas	10	200	$2.000,00
Armazenamento	1 GB	*free*	*free*

Preços *Azure AppFabric*

Item	Unidade	Free	Adicional
Largura de Banda de Entrada	Gigabyte	5 GB	$0,10
Largura de Banda de Saída	Gigabyte	5 GB	$0,15
CPU	Horas	25 horas	$0,12
Dados Armazenados	Gigabyte	500 MB	$0,15
Transações de Armazenamento	10.000 transações	10.000	$0,01

Preços *AppEngine*

Item	Unidade	Free	Adicional
Largura de Banda de Entrada	Gigabyte	5 GB	$0,10
Largura de Banda de Saída	Gigabyte	5 GB	$0,15
CPU	Horas	25 horas	$0,12
Dados Armazenados	Gigabyte	500 MB	$0,15
Transações de Armazenamento	10.000 transações	10.000	$0,01

Preços Heroku

Banco de Dados

Tipo	Tamanho	Preço
Compartilhado	5MB	free
Compartilhado	20GB	$15/mês
Dedicado	2TB, 1 CPU	$200/mês
Dedicado	2TB, 5 CPU	$400/mês
Dedicado	2TB, 20 CPU	$1600/mês

Dynos

Quantidade	Preço
1	free
2	$0,05/hora
5	$0,20/hora
10	$0,45/hora
20	$0,95/hora

Add-ons

Tipo	Configuração	Preço
Amanon RDS	-	*free*
Cron	Execução diária	*free*
Cron	Execução p/ hora	$3,00/mês
SSL	Piggyback	*free*
SSL	SNI	$5,00/mês
SSL	Custom	$100,00/mês
Domínio Customizado	Basic	*free*
Domínio Customizado	Wild Card	$5,00/mês

Fig. 6. Modelo de cobrança e preços: *force.com*[13], *Heroku*[8], *AppFabric*[12] e *AppEngine*[6]

A plataforma *force.com* possui a característica de escalabilidade automática, que é importante no modelo *pay-as-you-go* na medida em que proporciona disponibilidade e transmite a sensação de "recursos infinitos". Mas o seu modelo de cobrança baseado em nro. de usuários e nro. de aplicações e tabelas está distante do modelo *pay-as-you-go* da energia elétrica, ficando mais próximo dos mecanismos tradicionais de software baseado em licença.

O modelo do *Heroku* apresenta algumas características *pay-as-you-go* onde é possível selecionar configurações de software e hardware diferentes para suportar as aplicações, inclusive através de funcionalidades específicas chamadas *add-ons*. Entretanto, além da necessidade de se contratar uma infra-estrutura pré-alocada, esta infra-estrutura não escala automaticamente, o que transfere para o desenvolvedor do software a responsabilidade de contratar mais hardware quando julgar que o desempenho da sua aplicação não está satisfatório.

Já o *AppFabric* oferece um modelo mais próximo do *pay-as-you-go* , apresentando preços na casa de centavos de dólar/hora para itens como GB de dados armazenados, GB de dados transferidos e horas de CPU utilizadas. As horas de CPU, entretanto, são faturadas com base na disponibilidade de uso e não na utilização real. Além disso, no *AppFabric* a escalabilidade é manual, o que transfere ao desenvolvedor a responsabilidade de contratar mais CPUs quando julgar necessário. Assim, apesar de praticar preços na casa de centavos de dólar/hora, estas duas limitações fazem com que o *AppFabric* não

esteja tão próximo do modelo *pay-as-you-go* que estamos considerando.

O *AppEngine* é a plataforma que mais se aproxima do modelo de cobrança *pay-as-you-go* que utilizamos para consumir energia elétrica. Os preços são na casa de centavos de dólar/hora, mas sem as restrições existentes no *AppFabric*. No *AppEngine* também são cobrados GB armazenados, GB transferidos e horas de CPU, com a diferença de considerar somente as horas de CPU efetivamente utilizadas pela aplicação. Somando isto à escalabilidade automática, que aloca (e desaloca) máquinas conforme a necessidade da aplicação, temos um modelo *pay-as-you-go* praticamente igual ao do fornecimento de energia elétrica. A figura 7 faz uma analogia entre os modelos de cobrança das quatro plataformas e a sua aplicação hipotética ao fornecimento de energia elétrica.

Plataforma	Modelo de Cobrança	Analogia com Energia Elétrica
Force.com	Nº de usuários, Nº de aplicações e tabelas	Me diga **quantas pessoas** vão morar na sua casa e **quais os eletrodomésticos** que você tem e eu digo qual **o valor da sua conta**
Heroku	Infraesturutra alocada + *"add-ons"*	Fornecemos **geradores com várias configurações diferentes**. Você também pode **contratar serviços adicionais**, como rede estabilizada e no-breaks pagando um valor adicional por isso.
Azure AppFabric	Infraestrutura alocada + "dados sob demanda"	Temos **geradores de várias capacidades** a uma taxa fixa. **Além da taxa** cobramos **um valor proporcional** ao que você **utilizar**. Nos avise se você **precisar de mais energia do que a capacidade** do gerador, **alocamos um novo na hora**. Se quiser deixar de utilizá-lo nos avise para não cobrarmos.
App Engine	Infraestrutura e Dados "sob demanda"	Contrate o nosso serviço e comece a utilizar. **Utilize a energia livremente**. Nós vamos **medir a quantidade** que você utilizou **e enviar uma fatura** no final do mês

Fig. 7. Analogia das plataformas com o modelo *pay-as-you-go* de energia elétrica

4 Simulando impactos do desempenho no preço do SW

Conforme detalhado na seção 3, já há plataformas *Cloud* que adotam um modelo bem próximo ao que é utilizado atualmente para o fornecimento de energia elétrica, onde o preço a ser pago para utilizar uma determinada aplicação é função direta do volume de recursos que ela efetivamente consome. Estes recursos são precificados na casa de centavos de dólar e incluem itens como GB de dados armazenados, GB de dados transferidos e uso de CPU.

Para experimentar este conceito e termos uma noção real do seu impacto no desenvolvimento de software fizemos duas experiências. A primeira foi construir uma aplicação simples de consulta a base de dados em uma das plataformas identificadas e medir o impacto que uma otimização pontual de desempenho teria no preço final a ser pago pela aplicação. Escolhemos a plataforma *App Engine* para realizar esta experiência por ela apresentar um modelo mais próximo ao utilizado para consumo de energia elétrica. A segunda experiência foi modificar o código-fonte de um ERP para monitorar o volume de utilização dos serviços da sua camada servidora, estabelecendo um preço arbitrário para a execução de cada serviço. Para a segunda experiência usamos o ERP SIE [3] , utilizado pela UFSM e mais de 20 instituições brasileiras.

[3] O SIE é um sistema completo para administração de universidades, que engloba módulos como Almoxarifado, Compras, Recursos Humanos, Controle Orçamentário, Biblioteca, Controle Acadêmico, Patrimônio, Frota, Protocolo entre outros.

Experiência 1: Impacto financeiro do uso de CPU no *App Engine*

A aplicação desenvolvida para a primeira experiência foi uma operação bastante comum no desenvolvimento de software: a realização de uma consulta para recuperar todos os registros de uma determinada tabela do banco de dados. A aplicação foi feita utilizando o SDK Java do Google App Engine e a tabela foi criada através das suas APIs, contendo 4 colunas e 100 registros.

A mesma aplicação foi construída de duas formas diferentes. A primeira realizava um acesso direto a tabela para recuperar todos os seus registros usando a especificação JPA. A segunda fazia este acesso uma única vez e guardava o resultado em uma *cache*, de forma que acessos subsquentes consultam a *cache* ao invés de acessar a base de dados. Para esta implementação utilizamos a especificação JCache. Os dados referentes ao uso de CPU foram monitorados através de uma ferramenta do *App Engine*. Realizamos 6.000 requisições em cada aplicação, acompanhando o consumo efetivo de CPU em intervalos pré-determinados e o resultado está demonstrado na tabela da figura 8.

Tabela de Medições

Nro. de Requisições	Aplicação Monitorada	Chamadas de Acesso a Dados	Tempo de CPU (Total)	Tempo de CPU (Datastore)	Custo em US	Custo em RS
1000	JCACHE	328	0,02	0	0,002	0,0034
1000	JPA	1639	0,1	0,07	0,01	0,017
2000	JCACHE	328	0,02	0	0,002	0,0034
2000	JPA	3025	0,18	0,15	0,018	0,0306
4000	JCACHE	328	0,03	0	0,003	0,0051
4000	JPA	5797	0,34	0,3	0,034	0,0578
6000	JCACHE	328	0,04	0	0,004	0,0068
6000	JPA	8489	0,5	0,44	0,05	0,085

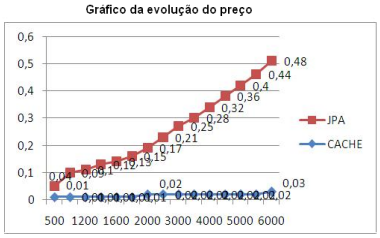

Fig. 8. Tabela e gráfico com medições da experiência 1: impacto financeiro da CPU no App Engine

Analisando a tabela percebe-se que as chamadas de acesso a dados permanecem constantes para a aplicação com JCache, enquanto aumentam proporcionalmente para a aplicação JPA. Isto se explica justamente pelo uso da *cache*. Também se nota que o tempo de CPU consumido exclusivamente para acesso a dados (*Datastore*) é significativo se comparado com o tempo total. No gráfico ao lado da tabela fica claro o impacto do custo nas aplicações. Enquanto o preço da aplicação JCache aumenta em torno de R$0,01 a cada 2.500 requisições, na aplicação JPA a evolução é de R$0,04 a cada 500 requisições.

Experiência 2: Monitoramento do uso do ERP SIE na UFSM

Para a segunda experiência fizemos modificações no código-fonte de um ERP para registrar o tempo gasto na execução de cada um dos serviços (métodos) publicados em sua camada servidora. Foi estabelecido um preço arbitrário para cada método, simulando um mecanismo de precificação com base em uso do serviço. Escolhemos o ERP SIE por ele ser utilizado pela UFSM, o que nos permite fazer simulações reais de volume de uso. O SIE foi desenvolvido em meados de 1998 utilizando a linguagem Delphi e uma arquitetura multi-camadas onde as regras de negócio são processadas através de chamadas RPC em um ou mais servidores de aplicação centralizados[3]. Modificamos o ERP para registrar o *log* de todas chamadas RPC e monitoramos o uso de um

dos servidores da UFSM durante 10 minutos. A figura 9 resume o resultado deste monitoramento. O método *ISGCA.GetRotulo*, por exemplo foi chamado 6.252 vezes neste período e teve um tempo total de resposta equivalente a 71.225 ms. Já o método *IRenovacao.GetRenovacoes* foi chamado 418 vezes, com um tempo de resposta total de 61.101 ms

TEMPO_TOTAL	NUM_CHAMADAS	NOME_METODO
71225	6252	ISGCA.GetRotulo
37887	5888	ICurriculoAluno.IndexInFields
6087	2544	IMascara.SubstituiMascara
13845	969	ISGCA.CtxCacheGetData
8992	935	IDisciplinasCurric.GetDisciplinaNoCurricDoCurso
373	744	ISGCA.GetDateTime
22855	702	IParInstituicao.GetRecords
20494	689	IConfiguracao.GetConfiguracao
6220	525	IEmprestimo.GetClassEmprestVinculada
10650	491	ISubCamposRegist.GetSubCamposRegist
8488	434	IDominioAut.GetDescrIdReferencia
3083	424	IUsuario.GetUsuarioConfig
61101	418	IRenovacao.GetRenovacoes
2096	248	IDocumento.EstaRequisitado
2531	219	IEmprestimo.GetDadosEmprestimo

TEMPO_TOTAL	NUM_CHAMADAS	NOME_METODO
24829	33	IRegistro.LoadRecord
305	33	IOcorMatricula.TurmaComAlunosMatriculados
5452	33	ICampFixRegistro.GetCampFixRegistros
279	33	ITurmaEspaco.GetEspacosTurma
389	33	ITurmaEspaco.GetHorariosSemEspFisTurma
138	33	IAlunoDocente.GetDocentesAlunos
1153	32	IFonteCorresp.InsertFonteCorresp
755	32	IRenovacao.Insert
646	32	IOrgInstituicao.GetTabEstruturada
371	32	IRenovacao.GetRecordPK
28183	32	IEmprestimo.Renovar
1376	32	ICorrespondencia.InsertCorrespondencia
232	32	IRenovacao.FieldsCount
11295	27	IDocOcorCurric.GetItensDocOcorCurric
954	27	IParAcadCurso.GetAnoPeriodoMatricula

Fig. 9. Volume de utilização e tempo de resposta dos métodos do SIE

Para fins de simulação, estabelecemos um preço arbitrário de 0,01 para cada chamada de método. O resultado está na figura 10, onde observamos que o método *IRenovacao.GetRenovacoes*, mesmo com um tempo total de resposta próximo ao *ISGCA.GetRotulo* (61.101 x 71.225 ms) tem preço final bastante inferior (R$4,18 x R$62,52). Caso semelhante ocorre em outros métodos, como *IDocOcorCurric.GetItensDocOcorCurric*, que teria um preço de apenas R$0,27 e tempo de resposta total de 11.295 ms. Como exemplo oposto, o método *IParInstituicao.GetRecords* teve 702 chamadas e um tempo total de 22.855 ms. Embora o tempo de resposta possa sofrer influência de outras variáveis (por exemplo, a disponibilidade do BD), de uma forma geral um tempo de resposta maior está relacionado a um maior consumo de recursos. Os dados coletados nesta experiência indicam que utilizar um preço padrão único para todos os métodos não é compatível com o modelo *pay-as-you-go*.

PRECO	TEMPO_TOTAL	CHAMADAS	NOME_METODO
R$ 62,52	71225	6252	ISGCA.GetRotulo
R$ 58,88	37887	5888	ICurriculoAluno.IndexInFields
R$ 25,44	6087	2544	IMascara.SubstituiMascara
R$ 9,69	13845	969	ISGCA.CtxCacheGetData
R$ 9,35	8992	935	IDisciplinasCurric.GetDisciplinaNoCurricDoCurso
R$ 7,44	373	744	ISGCA.GetDateTime
R$ 7,02	22855	702	IParInstituicao.GetRecords
R$ 6,89	20494	689	IConfiguracao.GetConfiguracao
R$ 5,25	6220	525	IEmprestimo.GetClassEmprestVinculada
R$ 4,91	10650	491	ISubCamposRegist.GetSubCamposRegist
R$ 4,34	8488	434	IDominioAut.GetDescrIdReferencia
R$ 4,24	3083	424	IUsuario.GetUsuarioConfig
R$ 4,18	61101	418	IRenovacao.GetRenovacoes
R$ 2,48	2096	248	IDocumento.EstaRequisitado
R$ 2,19	2531	219	IEmprestimo.GetDadosEmprestimo

PRECO	TEMPO_TOTAL	CHAMADAS	NOME_METODO
R$ 0,33	24829	33	IRegistro.LoadRecord
R$ 0,33	305	33	IOcorMatricula.TurmaComAlunosMatriculados
R$ 0,33	5452	33	ICampFixRegistro.GetCampFixRegistros
R$ 0,33	279	33	ITurmaEspaco.GetEspacosTurma
R$ 0,33	389	33	ITurmaEspaco.GetHorariosSemEspFisTurma
R$ 0,33	138	33	IAlunoDocente.GetDocentesAlunos
R$ 0,32	1153	32	IFonteCorresp.InsertFonteCorresp
R$ 0,32	755	32	IRenovacao.Insert
R$ 0,32	646	32	IOrgInstituicao.GetTabEstruturada
R$ 0,32	371	32	IRenovacao.GetRecordPK
R$ 0,32	28183	32	IEmprestimo.Renovar
R$ 0,32	1376	32	ICorrespondencia.InsertCorrespondencia
R$ 0,32	232	32	IRenovacao.FieldsCount
R$ 0,27	11295	27	IDocOcorCurric.GetItensDocOcorCurric
R$ 0,27	954	27	IParAcadCurso.GetAnoPeriodoMatricula

Fig. 10. Tempo de processamento e simulação de preço para uso dos métodos do SIE

5 Impacto no desenvolvimento e na precificação de SW

A partir da avaliação das *PaaS* e nas experiências que realizamos é possível projetar um cenário onde os desenvolvedores de software podem construir os

seus sistemas e colocá-los a disposição dos usuários em uma plataforma *Cloud* com modelo *pay-as-you-go*, onde o preço é proporcional ao volume de recursos utilizado. Nesta hipótese, quanto maior a utilização do sistema maior será o preço a ser pago ao fornecedor. Uma das questões que surgem neste cenário é: quem irá pagar pelos recursos da plataforma que serão consumidos pelo sistema, o desenvolvedor/distribuidor do sistema ou o cliente?

Nossa experiência, adquirida em mais de 10 anos de atividades relacionadas ao desenvolvimento, manutenção e distribuição do ERP SIE, nos diz que no caso de sistemas ERP tradicionais esta conta é paga pelos clientes. O distribuidor do ERP oferece as licenças do sistema e um pacote de treinamentos, especificando ao cliente os recursos de hardware que devem ser adquiridos para suportar a operação do sistema. Entretanto, as plataformas *Cloud* com modelo *pay-as-you-go* afetam este cenário, fortalecendo a possibilidade de incorporação do custo dos recursos de hardware ao preço final do software.

Por si só, a incorporação destes recursos ao preço final do software não é uma novidade, já que atualmente os sistemas podem ser hospedados em *Data Centers* e acessados pelos clientes através de uma conexão internet. A novidade agora é que somente os recursos efetivamente utilizados pelo sistema geram custos. No modelo anterior um desenvolvedor de software poderia contratar um *Data Center*, hospedar a sua aplicação e estabelecer um preço aos seus clientes. Conforme a utilização do sistema pelos usuários, os computadores contratados poderiam ficar ociosos ou sobrecarregados, mas o preço pago pelo desevolvedor ao *Data Center* permanecia o mesmo. No modelo novo o preço pago ao fornecedor da plataforma varia conforme a utilização do sistema pelos usuários e o consequente volume de recursos consumidos.

Por outro lado o volume de recursos consumidos por um sistema não é função exclusiva da variação da demanda gerada por seus usuários, mas também do quanto o código-fonte está otimizado para consumir o menor volume de recursos possível. As experiências da seção 4 nos dão um exemplo desta situação. Neste cenário a construção de código otimizado ganha importância e alguns aspectos do desenvolvimento de software podem ser modificados.

As análises que fazemos a seguir consideram a hipótese de que i) o software é desenvolvido e distribuído através de uma plataforma *Cloud* ; ii) a plataforma utiliza o modelo de cobrança *pay-as-you-go*, baseado no volume de utilização de recursos; iii) os custos dos recursos consumidos pelo software são assumidos pelo seu desenvolver/ditribuidor e incorporados ao preço final; e iv) o preço final não é uma função direta do volume de recursos consumidos.

5.1 Impacto no desenvolvimento de software

A primeira mudança é também a mais evidente e está relacionada à própria existência de **funcionalidades** que, **por não estarem devidamente**

otimizadas consomem um volume de recursos maior do que poderiam, como é o caso da experiência 2 da seção 4. Enquanto no modelo tradicional o impacto desta falta de otimização dificilmente afetará o desenvolvedor do software, no modelo *pay-as-you-go* ela irá consumir uma fatia do seu lucro, na medida em que ele pagará um valor maior ao fornecedor da plataforma.

É claro que o custo final de um código não otimizado irá depender do **volume de utilização da funcionalidade** propriamente dita, e este é um segundo aspecto em que o novo modelo de cobrança afeta o desenvolvimento de software. Embora tenhamos a prática de identificar possíveis gargalos de desempenho e otimizá-los o máximo possível de forma a garantir um bom tempo de resposta, o foco desta otimização agora passa a ser diferente. No novo modelo, além destas funcionalidades que são gargalos, também passam a ter grande importânca aquelas que são utilizadas de forma rotineira pelos usuários, mas que possuem uma frequência de uso significativa. No caso do SIE, por exemplo, a funcionalidade que recupera todas as unidades e departamentos da universidade é utilizada pela grande maioria dos usuários pelo fato de estar presente em várias aplicações diferentes. Apesar de não ter impacto na disponibilidade do sistema, uma otimização no código que implementa esta funcionalidade pode ter um reflexo importante no consumo total de recursos do sistema. Um exemplo deste comportamento pode ser verificado na figura 10, onde o método *IParInstituicao.GetRecords* aparece como sendo um dos mais executados. Outros métodos, como *ISGCA.GetRotulo* ou *ICurriculoAluno.IndexInFields* apresentam o mesmo perfil de comportamento. Desta forma, o escopo da preocupação com a otimização de código passa a ser maior, visto que melhorias no desempenho de métodos como os descritos acima tendem a aumentar o lucro do desenvolvedor/distribuidor do software.

Dentro deste contexto, a avaliação do volume de acessos que um determinado método ou funcionalidade terá passa a ser uma prática recomendada durante o desenvolvimento da aplicação, como forma de identificar quais trechos de código devem ter um maior ou menor cuidado com otimização. Ainda assim, novos pontos de otimização poderão ser descobertos durante a utilização efetiva do sistema pelos usuários, uma vez que **cada cliente pode** utilizar o sistema de maneira diferente e **dar preferência a outras funcionalidades que não aquelas previstas** pelos desenvolvedores do software.

Outro aspecto recorrente constatado em nossa experiência com o SIE é a solicitação, por parte do cliente, da **adição de novas funcionalidades** ou mesmo de aplicações ou sistemas inteiros à solução. O procedimento normal neste caso é fazer a estimativa do número de horas que serão necessárias e aprovar um orçamento com o cliente para que a funcionalidade seja desenvolvida. Com o modelo *pay-as-you-go* outros dois aspectos podem ser afetados: i) como será preciso avaliar (e/ou implementar) o grau de otimização necessário a partir da expectativa do volume de uso das novas funcionalidades,

é possível que os orçamentos para desenvolver a funcionalidade tenham seu preço elevado; ii) o aumento das funcionalidades do sistema tende a aumentar sua utilização como um todo, o que poderá implicar um maior consumo de recursos, refletindo no valor pago ao fornecedor da plataforma e com consequente redução no lucro do desenvolvedor/distribuidor do sistema.

Além de **valorizar o trabalho relacionado à otimização de código**, esta preocupção com o uso racional de recursos durante a etapa de construção do software também pode ter impacto nas técnicas utilizadas para **análise de requisitos**, uma vez que a questão envolvendo o volume de utilização de uma determinada funcionaliade passa a ser um ponto importante a ser levantado já nas fases iniciais do desenvolvimento do software.

Outra questão afetada diz respeito ao **póprio foco da otimização de código**. No modelo tradicional os problemas de otimização costumam ter o objetivo de minimizar tanto quanto possível o tempo de resposta de uma funcionalidade e aumentar o seu *throughput*. Com o modelo *pay-as-you-go* o problema a ser resolvido passa a ser formulado de maneira diferente: o objetivo será **utilizar o mínimo de recursos possível** para atingir um **tempo de resposta aceitável** para o usuário. Esta mudança é justamente em função de dois conceitos básicos do modelo *pay-as-you-go*: pagar apenas pelos recursos utilizados e a sensação de "recursos infinitos" oferecida pelos fornecedores *PaaS*. No modelo tradicional lidávamos com uma limitação de hardware e procurávamos extrair dele o máximo possível. No novo modelo o hardware é ilimitado e devemos utilizar somente o que for estritamente necessário.

5.2 Impacto na precificação do software:

Aspectos de preço e licenciamento, assim como a escalabilidade estão entre os desafios apontados pelo trabalho da Universidade de Berkeley[11] e várias pesquisas vem sendo realizadas nos últimos anos tratando destes temas. No caso de *IaaS*, os modelos de cobrança vem convergindo para a precificação na casa de centavos de dólar para itens como horas de CPU, GB armazenados por mês GB de dados transferidos. Esta convergência pode ser constatada no site dos próprios fornecedores e também no trabalho de Shifeng Shang[14], que além de comparar preços entre alguns dos principais fornecedores, apresenta a proposta de tratar o mercado *Cloud* como um conjunto de recursos disponíveis que podem ser comercializados em um mercado semelhante ao de *commodities*. Há também estudos com modelos de preço baseados em *cache*[9] e modelos para alocar recursos com base em SLAs, levando em conta valores e multas contratados com os clientes para decidir sobre a quebra ou não de um SLA[10].

Estes estudos vem potencializando a capacidade de alocação e escalabilidade dos fornecedores *Cloud*, e apesar de estarem voltados principalmente para o modelo *IaaS*, eles também irão benefeciar os fornecedores *PaaS*. En-

tretanto, a maioria dos estudos sobre modelos de preço *Cloud* está focada no fornecedor. Há poucos trabalhos abordando este problema sob a ótica do desenvolvedor de software, que em última instância utilizará as plataformas para distribuir as suas soluções. O trabalho de Evan Asfoura[2], apesar de não ter relação direta com ambientes *Cloud*, propõe um modelo de preço para um sistema de *workflow* que é aderente ao conceito *pay-as-you-go*. O modelo é baseado na complexidade do fluxo a ser executado: quanto maior o número de atividades e a complexidade das decisões que ele precisar tomar, maior será o seu preço. Este modelo pode ser uma alternativa para sistemas de *workflow*, mas não trata de outros tipos de software, os quais possuem funcionalidades com características diversas. O ERP SIE é um exemplo onde existem funcionalidades que não seriam bem representadas desta forma, como é o caso da funcionalidade que lista as unidades e departamentos da universidade (*IParInstituicao.GetRecords*, na seção 4).

O ensaio que apresentamos a seguir leva em consideração os impactos que discutimos na seção 5.1, onde destacamos, dentre outros, os seguintes aspectos que poderão influenciar o lucro dos desenvolvedores/distribuidores de software: i) o volume de recursos consumido por uma dada funcionalidade; ii) o volume de uso de cada funcionalidade.

Para cobrir estes dois aspectos, o modelo que estamos desenvolvendo estimula que o desenvolvedor do software estabeleça um preço unitário para cada funcionalidade do seu sistema, de forma a garantir uma margem de lucro sempre que ela for usada, juntando a esta prática um mecanismo de monitoramento que permita acompanhar em tempo real o lucro (ou prejuízo) de cada funcionalidade (figura11). O mecanismo de monitoramento é importante pois, ainda que as funcionalidades tenham sua expectativa de uso estimada, esta estimativa é suscetível a erros e ainda está sujeita às variações no comportamento dos usuários com relação ao sistema. Além disso o modelo sugere que, durante o levantamento de requisitos do software sejam identificadas não apenas quais funcionalidades deverão existir, mas também a frequência com que se espera que elas sejam utilizadas.

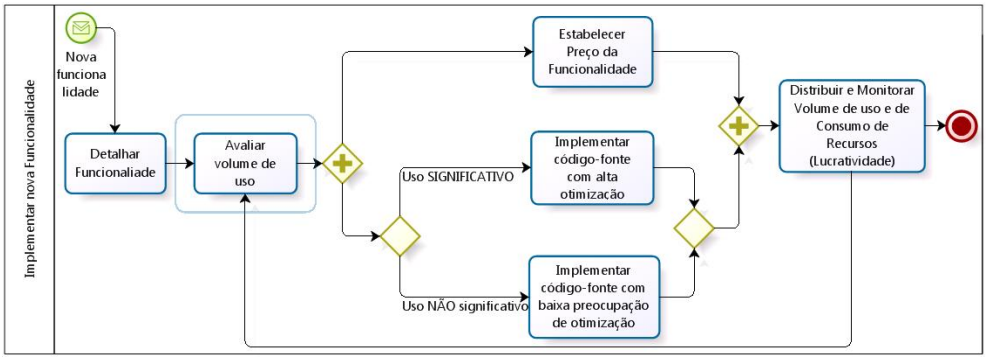

Fig. 11. Esquema para o desenvolvimento de software em PaaS com modelo *pay-as-you-go*

Para implementar este modelo é necessário uma arquitetura de software que permita monitorar o uso de cada funcionalidade do sistema e a quantidade de recursos que elas estão consumindo individualmente em uma granularidade que pode estar em nível de aplicação, botões e menus, de serviços da camada servidora, etc. (quanto menor a granularidade, maior o nível de controle que teremos sobre a lucratividade do sistema). Para simular os resultados que podem ser obtidos com esta abordagem fizemos uma nova experiência com o SIE, adicionando ao mecanismo de *log* novas funções para monitorar cada um dos métodos que compõem a sua camada servidora quanto a: i) o volume de uso de cada um; ii) o tempo de resposta individual; iii) qual aplicação está sendo utilizada para chamar o método e iv) qual usuário estava utilizando a aplicação. Monitoramos o sistema por 10 minutos mantendo o preço unitário de 0,01 por método e os resultados estao na figura 12.

Preço por Aplicação

PRECO	NOME_EXEC
R$ 70,48	ACMAAdaptaCurric.exe
R$ 29,27	ACMOcorCurric.exe
R$ 19,14	BibCRegistroMarc.exe
R$ 16,27	EMEmpenho.exe
R$ 12,30	ACMBolsista.exe
R$ 10,91	GCANavegacao.exe
R$ 9,95	ACMOfertaTurmasPorCurso.exe
R$ 9,53	CEENTEstoque.exe
R$ 5,88	RHCFuncionarios.exe
R$ 5,59	ACMBolsistasPorUnidade.exe
R$ 4,62	CESolProdutos.exe
R$ 4,46	OTrocaClassDesp.exe
R$ 3,55	BibEControlePortaria.exe
R$ 2,56	TRAProcesso.exe
R$ 2,35	ACMBolsa.exe

Preço por Centro de Custo

PRECO	CENTRO_CUSTO
R$ 114,20	01.25.00.00.0.0 - DEPARTAMENTO DE REGISTRO E CONTROLE ACADÊMICO - DERCA
R$ 24,57	01.18.00.00.0.0 - DEPTO. MATERIAL E PATRIMÔNIO - DEMAPA
R$ 22,52	01.06.00.00.0.0 - PRÓ-REITORIA DE ASSUNTOS ESTUDANTIS - PRAE
R$ 20,10	01.30.00.00.0.0 - BIBLIOTECA CENTRAL - BC
R$ 14,18	01.21.00.00.0.0 - PRÓ-REITORIA DE RECURSOS HUMANOS - PRRH
R$ 12,72	01.22.00.00.0.0 - DEPTO. REGISTRO E CONTROLE ACADÊMICO - DERCA
R$ 3,83	01.05.00.00.0.0 - PRÓ-REITORIA DE ADMINISTRAÇÃO - PRA
R$ 3,13	01.10.00.00.0.0 - PRÓ-REITORIA DE PLANEJAMENTO - PROPLAN
R$ 2,29	01.01.00.00.0.0 - GABINETE DO REITOR
R$ 1,76	09.00.00.00.0.0 - CENTRO DE EDUCAÇÃO FÍSICA E DESPORTOS - CEFD
R$ 1,68	04.00.00.00.0.0 - CENTRO DE CIÊNCIAS DA SAÚDE - CCS
R$ 1,36	01.09.00.00.0.0 - PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA - PRPGP
R$ 1,11	01.84.00.00.0.0 - DEPARTAMENTO DE ARQUIVO GERAL - DAG
R$ 1,04	01.12.00.00.0.0 - DEPARTAMENTO DE CONTABILIDADE E FINANÇAS - DCF
R$ 0,71	01.11.00.00.0.0 - PRÓ-REITORIA DE INFRAESTRUTURA

Fig. 12. Resultados de uma simulação do modelo com o ERP SIE

As pequenas mudanças introduzidas no mecanismo de *log* nos permitiram identificar, a partir da precificação unitária dos métodos, qual preço o usário final (no caso, a UFSM) pagaria por cada aplicação do sistema. Outro resultado interessante que obtivemos está relacionado ao Centro de Custo [4]. Como o SIE é um ERP que contém informações integradas sobre vários setores da universidade, ao registrar no *log* quais usuários estão realizando cada uma das chamadas é possível recuperar o Centro de Custo a que eles estão associados. Para um cliente de sistemas do tipo ERP esta informação pode ser valiosa, pois a partir dela é possível apropriar o custo financeiro que cada unidade ou departamento está gerando para a instituição com o uso do sistema. Extrapolando esta visão do ponto de vista do cliente, também seria possível identificar quanto cada usuário está gastando individualmente com o uso do sistema e ainda utilizar este tipo de informação como insumo para otimizar os processos de trabalho para que consumam recursos de TI da melhor forma possível.

[4] Uma unidade usada p/ apropriar despesas, em geral relacionada a contabilidade de custos.

6 Conclusão

Neste trabalho realizamos uma análise do modelo *Cloud Computing* e seus desafios, dando a atenção especial às soluções do tipo Plataforma como Serviço (*PaaS*). Identificamos as plataformas disponíveis atualmente e fizemos uma avaliação de algumas delas sob a ótica do modelo de cobrança *pay-as-you-go* que estamos habituados a utilizar para consumir energia elétrica. Analisamos o impacto que as plataformas *Cloud* com modelo *pay-as-you-go* podem gerar no desenvolvimento de software tomando como base a experiência de mais de 10 anos que temos com o desenvolvimento e distribuição do ERP SIE.

Nesta análise identificamos que as questões relacionadas à **otimização de código** deverão ter uma importância diferenciada no desenvolvimento de sistemas, uma vez que neste modelo os recursos de hardware são consumidos e faturados sob demanda. O mecanismo *pay-as-you-go* tem impacto também na forma que distribuímos o software, já que é bastante possível que os custos com os recursos de hardware passem a ser incorporados ao software, o que pode impactar as **estratégias de precificação**. Identificamos ainda uma **mudança no foco das otimizações** de código, que passariam a se concentrar em minimizar os recursos consumidos ao invés do tradicional "diminuir o tempo de resposta" e maximizar o *throughput*.

Dentro deste contexto demos início ao estudo de um modelo de desenvolvimento de software que contemple as mudanças que identificamos. O modelo está em um estágio inicial e o seu objetivo principal é destacar a importância que as atividades de otimização de código passam a ter quando trabalhamos com plataformas *Cloud pay-as-you-go*, alinhando isso com o problema da precificação. Simulamos este modelo na utilização do SIE na UFSM e a partir desta simulação identificamos oportunidades tanto para os desenvolvedores de software (estimar e monitorar o lucro de cada funcionalidade do sistema) quanto para os clientes (identificar e apropriar o custo da utilização de recursos de TI de forma individualizada).

As mudanças originadas pelo uso do modelo *pay-as-you-go* nas plataformas *Cloud* ainda são pouco exploradas, e acreditamos que o campo de pesquisa é bastante amplo. Na área de desenvolvimento de software muitos estudos ainda podem ser realizados a cerca das metodologias de desenvolvimento e das disciplinas de engenharia de software. Mudanças em *frameworks* e até mesmo nas próprias funcionalidades fornecidas pelas plataformas *Cloud* também podem ser objeto de estudos futuros. Os modelos de preço para fornecimento e manutenção de software no modelo *Cloud* ainda estão na infância se comparados com os próprios mecanismos que já vem sendo utilizados pelas *IaaS*.

Outro aspecto a ser estudado é a própria forma como os clientes utilizam os seus softwares. As empresas desenvolveram durante os anos vários métodos para racionalizar o uso de recursos com o intuito de reduzir custos e ganhar

competitividade. Metodologias de análise e otimização de processos são bastante utilizadas neste tipo de trabalho e podem ser avaliadas como forma de identificar e eliminar desperdícios com o uso de software, já que, conforme concluímos, no modelo *pay-as-you-go* temos a possibilidade de identificar os custos de TI de forma individualizada.

O tema *Cloud Computing* ainda é novo e há vários estudos para serem realizados. Tanto a análise que fizemos quanto o modelo que estamos desenvolvendo podem ser melhorados e aprofundados para alcançar outras áreas e nossa expectativa que este trabalho contribua para que novas pesquisas sejam feitas sobre o tema.

Referências

[1] Amazon, *Amazon s3*, http://aws.amazon.com/s3, acessado em Dez/2010.

[2] Asfoura, E., N. Jamous, G. Kassem and R. Dumke, *Pricing-model for marketing of ferp workflow as product*, in: *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, 2010, pp. 321 –325.

[3] Barbosa, F. P., *Projeto e implementação de um framework para desenvolvimento de aplicações em três camadas*, Technical report, Curso de Ciência da Computação. Universidade Federal de Santa Maria., Santa Maria (2000).

[4] Buyya, R., C. S. Yeo and S. Venugopal, *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities*, in: *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, 2008, pp. 5 –13.

[5] Foster, I., Y. Zhao, I. Raicu and S. Lu, *Cloud computing and grid computing 360-degree compared*, in: *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.

[6] Google, *Appengine*, http://code.google.com/appengine/docs/billing.html, acessado em Dez/2010.

[7] Hamid R Motahari-Nezhad, S. S., Bryan Stephenson, *Outsourcing business to cloud computing services: Opportunities and challenges*, Technical report, USA (2009).
URL http://www.hpl.hp.com/techreports/2009/HPL-2009-23.pdf

[8] Heroku, *Heroku*, http://www.heroku.com/pricing, acessado em Dez/2010.

[9] Kantere, V., D. Dash, G. Francois, S. Kyriakopoulou and A. Ailamaki, *Optimal service pricing for a cloud cache*, Knowledge and Data Engineering, IEEE Transactions on **PP** (2011), p. 1.

[10] Maci Andas, M., J. Fito and and J. Guitart, *Rule-based sla management for revenue maximisation in cloud computing markets*, in: *Network and Service Management (CNSM), 2010 International Conference on*, 2010, pp. 354 –357.

[11] Michael Armbrust, R. G. A. D. J. R. H. K. A. K. G. L. D. A. P. A. R. I. S. M. Z., Armando Fox, *Above the clouds: A berkeley view of cloud computing*, Technical report, Berkeley, California, USA (2009).
URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[12] Microsoft, *Azure appfabric*, http://www.microsoft.com/windowsazure/appfabric/overview/, acessado em Dez/2010.

[13] Salesforce, *Salesforce platform*, http://www.salesforce.com/platform/platform-edition/, acessado em Dez/2010.

[14] Shang, S., J. Jiang, Y. Wu, G. Yang and W. Zheng, *A knowledge-based continuous double auction model for cloud market*, in: *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, 2010, pp. 129 –134.

[15] Wikipedia, *Wikipedia categories: Cloud platform*, http://en.wikipedia.org/wiki/Category:Cloud_platforms, acessado em Dez/2010.