國立雲林科技大學
100學年度博士班招生考試試題

系所別：資訊管理系
科 目：資訊管理文獻

Based on the abridged paper below, please answer the following questions:

1. Please detailed explain the difference between user satisfaction and technology acceptance by this paper and your opinions. (20%)
2. From you have learned research method, please tell me the main meaning of Table 2. (10%)
3. After reading the paper, please practically point out two future works in application of information system.(20%)
4. Use examples to describe and distinguish between object-based beliefs, attitudes and behavioral beliefs, attitudes, then relate them to IS design, implementation, and prediction of usage. (15%)
5. Describe the correspondence principle, and comment on the role it plays to bridge the gap between two research streams. (15%)
6. Based on the major findings of this research, suggest implications for practice as well as for research. (20%)

Information Systems Research
Vol. 16, No. 1, March 2005, pp. 85–102
ISSN 1047-7047 | EISSN 1526-5536 | 05 | 1601 | 0085

DOI 10.1287/isre.1050.0042

A Theoretical Integration of User Satisfaction and Technology Acceptance

Barbara H. Wixom, Peter A. Todd
McIntire School of Commerce, University of Virginia, Charlottesville, Virginia 22904-4173
{bwixom@mindspring.com, ptodd@virginia.edu}

1. Introduction

Information technology (IT) researchers have developed rich streams of research that investigate the factors and processes that intervene between IT investments and the realization of their economic value. Commonly, researchers tie these factors and processes to user perceptions about IT and how it impacts their work. Although researchers have examined such perceptions in dozens of different ways (DeLone and McLean 1992), in general, there have been two dominant approaches employed—user satisfaction (e.g., Bailey and Pearson 1983, Ives et al. 1983, Melone 1990, Seddon 1997) and technology acceptance (e.g., Davis 1989, Hartwick and Barki 1994, Szajna 1996, Venkatesh et al. 2003). Both research streams offer valuable contributions to our understanding of IT, although each tells only part of the story. The purpose of this study is to integrate the two research streams so that, together, they can provide a more complete understanding of the way in which system features ultimately influence IT usage.

The user satisfaction literature explicitly enumerates system and information design attributes (e.g., information accuracy and system reliability), making it a potentially useful diagnostic for system design; however, user satisfaction is a weak predictor of system usage (Davis et al. 1989, Goodhue 1988, Hartwick and Barki 1994, Melone 1990). This is attributable to the fact that beliefs and attitudes about objects (such as an information system) are generally poor predictors of behaviors (such as system usage) (Ajzen and Fishbein, in press).

By contrast, the technology acceptance literature (i.e., the technology acceptance model, or TAM) provides sound predictions of usage by linking behaviors to attitudes and beliefs (ease of use and usefulness) that are consistent in time, target, and context with the behavior of interest (system usage). Despite its predictive ability, TAM provides only limited guidance about

how to influence usage through design and implementation (Taylor and Todd 1995, Venkatesh et al. 2003). For example, designers receive feedback regarding ease of use and usefulness in a general sense, but they do not receive actionable feedback about important aspects of the IT artifact itself (e.g., flexibility, integration, completeness of information, and information currency). Such guidance was a core objective in the development of TAM, but one that has received limited attention (Davis et al. 1989).

Although user satisfaction and technology acceptance have evolved largely as parallel research streams, the two approaches can and should be integrated (Goodhue 1988, Hartwick and Barki 1994, Melone 1990, Seddon 1997). Such integration can help build a conceptual bridge from design and implementation decisions to system characteristics to the prediction of usage. Ultimately, this would improve the predictive value of user satisfaction and augment the practical utility of technology acceptance. Furthermore, by theoretically integrating the two very important IT research streams, we can answer the call to provide a way for perception-based IT research to more fully examine the role of the IT artifact (Benbasat and Zmud 2003, Orlikowski and Iacono 2001).

To accomplish this, we apply concepts from the broader attitude literature (e.g., Ajzen 2001; Ajzen and Fishbein, in press; Eagly and Chaiken 1993; Fazio and Olson 2003; Haddock and Zanna 1999). Specifically, the paper develops a model that explicitly distinguishes the object-based beliefs and attitudes found in the user satisfaction literature from behavioral beliefs and attitudes in the technology acceptance literature. It enumerates a set of system and information characteristics that influence system and information quality, describes how they in turn influence object-based beliefs and attitudes with the system and the information it produces, and then describes how these object-based attitudes toward the system can shape the behavioral beliefs of usefulness, ease of use, and, ultimately, system usage.

The remainder of the paper proceeds as follows. Section 2 builds the theoretical arguments for the proposed research model. In §3, we present the background for a preliminary study that tested this model in the context of data warehousing. The results of an empirical test of this model are presented in §4. They are based on a sample of 465 users of data warehousing predefined reporting software from seven different organizations. Finally, in §5, we provide a discussion of the findings and an agenda for future research.

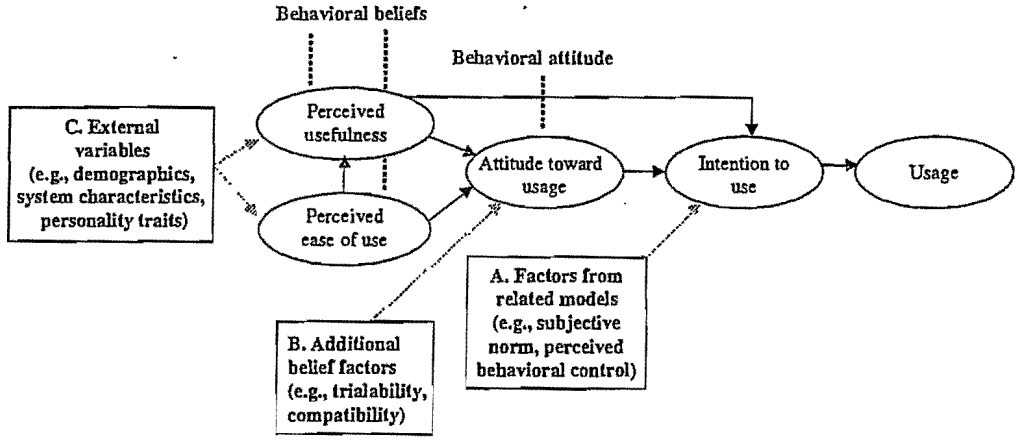
2. Theoretical Development

2.1. Understanding Behavioral Beliefs and Attitudes

According to the expectancy-value theory developed by Ajzen and Fishbein (1980), external variables influence beliefs about the outcomes associated with performing a behavior, which in turn shape attitudes toward performing a behavior. Attitude, in turn, influences intention to perform the behavior and, ultimately, influences the behavior itself. Satisfaction in a given situation is a person's feelings or attitudes toward a variety of factors affecting that situation. As articulated in the theory of reasoned action (TRA), these relationships will be predictive of behavior when the attitude and belief factors are specified in a manner consistent with the behavior to be explained in terms of time, target, and context (Ajzen and Fishbein, in press; Fazio and Olson 2003). Within the IT literature, these ideas have taken shape in the form of the TAM. TAM has been widely applied to understand the attitude one holds about the use of technology, which is used to predict the adoption and use of information technology. The attitude construct in TAM represents attitude toward the behavior of using technology.

Over the past decade, the technology acceptance literature has included a large number of empirical tests, comparisons, model variants, and model extensions. As Figure 1 illustrates, researchers have extended TAM in three primary ways to provide greater understanding and explanatory power and additional points of managerial leverage in its application. The first approach involves introducing factors from related models, such as subjective norm, perceived behavioral control, and self-efficacy (e.g., Hartwick and Barki 1994, Taylor and Todd 1995, Mathieson et al. 2001). A second approach involves introducing additional or alternative belief factors to the model. Most often, this includes adding key related factors from the diffusion of innovation literature, such as trialability, compatibility, visibility, or result demonstrability (Agarwal and Prasad 1997, Karahanna et al.

Figure 1 The TAM (Davis 1989) and Three Popular Extensions



1999, Plouffe et al. 2001). A third approach has been to examine external variables, which are antecedents to or that moderate the influence of ease of use and usefulness within the TAM, such as personality traits and demographic characteristics (e.g., Gefen and Straub 1997, Venkatesh 2000, Venkatesh and Morris 2000). Venkatesh et al. (2003) provide a comprehensive examination of eight different models and derive a unified theory of acceptance and use of technology.[1]

Despite this extensive research activity, only a handful of TAM studies have looked explicitly at the role of system characteristics as antecedents to ease of use or usefulness (e.g., Davis 1993, Igbaria et al. 1995, Lim and Benbasat 2000). For the most part, these studies have treated system characteristics at a holistic level or have looked at a limited number of features. One exception to this is the work by Hong et al. (2001–2002) that examines how dimensions of usability (information relevance, clarity of terminology, and screen design) influence ease of use and usefulness in the context of a digital library application. Their results show mixed effects with only relevance influencing both usefulness and ease of use. In their integration of the technology acceptance literature, Venkatesh et al. (2003) stress the need to extend this literature by explicitly considering system and information characteristics and the way in which they might influence the core beliefs in TAM, and might indirectly shape system usage.

2.2. Understanding Object-Based Beliefs and Attitudes

In contrast to the technology acceptance literature, system and information characteristics have been core elements in the literature on user satisfaction (DeLone and McLean 1992). Within this literature, user satisfaction is typically viewed as the attitude that a user has toward an information system; therefore, it represents an object-based attitude. User satisfaction primarily has been measured by various subsets of beliefs about specific systems, information, and other related characteristics (e.g., IT service).

This becomes clear when one examines user satisfaction instruments, such as Bailey and Pearson (1983), Baroudi and Orlikowski (1988), Doll and Torkzadeh (1988), and Ives et al. (1983) (see Table 1). These instruments use a characteristics-based approach for measuring user satisfaction. Although these instruments have been criticized for containing an arbitrary assortment of characteristics (Galletta and Lederer 1989), the items from user satisfaction instruments appear to conceptually represent a relatively small number of higher order constructs. Thus, the existing measures of user satisfaction provide a useful base for identifying and examining the underlying structure of system and information characteristics.

A fundamental problem with user satisfaction research has been its limited ability to predict system

[1] The Venkatesh et al. (2003) study provides an excellent review of TAM studies.

Table 1 Satisfaction Surveys and Constructs

External variables	Instrument characteristics	Bailey and Pearson (1983)	Ives et al. (1983)	Baroudi and Orlikowski (1988)	Doll and Torkzadeh (1988)
System quality	Accessibility	X	X		
	Timeliness	X	X		X
	Language	X	X		
	Flexibility	X	X		
	Integration	X	X		
	Efficient				X
Information quality	Accuracy	X	X	X	X
	Precision	X	X	X	X
	Reliability	X	X	X	X
	Currency	X	X		X
	Completeness	X	X	X	X
	Format	X			X
	Volume	X	X		
Service quality	Relationship with EDP staff	X	X	X	
	Communication with EDP staff	X	X	X	
	Technical competence of EDP staff	X	X		
	Attitude of EDP staff	X	X	X	
	Schedule of products or services	X	X		
	Time required for new development	X	X	X	
	Processing of change requests	X	X	X	
	Vendor support	X			
	Response time	X	X		
	Means of input with EDP center	X			
Usefulness	Usefulness	X	X		X
	Relevancy	X	X	X	X
Ease of use	User friendly				X
	Easy to use				X
Outcome expectations	Expectations	X	X		
	Understanding of systems	X	X	X	
	Confidence in the system	X	X		
	Feelings of participation	X	X	X	
	Feelings of control	X	X		
	Degree of training	X	X	X	
	Job effects	X	X		
Organizational factors	Top management involvement	X	X		
	Organizational competition with EDP	X			
	Priorities determination	X	X		
	Charge-back method	X			
	Error recovery	X	X		
	Security of data	X			
	Documentation	X	X		
	Organizational position of EDP	X	X		

Note. EDP = electronic data processing.

usage (Davis et al. 1989, DeLone and McLean 1992, Goodhue 1988, Hartwick and Barki 1994, Melone 1990, Seddon 1997). However, when one considers the general attitude literature, the equivocal relationship between user satisfaction and usage can be understood. For a belief or attitude to be directly predictive of behavior, it needs to be consistent in time, target, and context with the behavior. Therefore, satisfaction with the system and its information output is unlikely to be directly predictive of the use of that system.

Instead, user satisfaction needs to be recognized as an object-based attitude (Ajzen and Fishbein 1980, p. 84) whereby it serves as an external variable with influences on intention and behavior that are fully mediated by behavioral beliefs and attitudes (Ajzen and Fishbein 1980; Eagly and Chaiken 1993, p. 205). For example, one's satisfaction with the reliability of a system does not directly impact whether one will use the system. However, beliefs about reliability certainly will affect one's attitude toward the system, which will shape behavioral beliefs about using the system (e.g., ease of use). It is the system behavioral belief (ease of use) that directly influences attitude toward use and, ultimately, usage. In the user satisfaction literature, the mediating behavioral beliefs and attitudes are absent, and inattention to this conceptual gap explains the equivocal relationship between system satisfaction and system usage (see Figure 2).

Empirical evidence shows that object-based attitude is generally a weak predictor of behavior (Ajzen and Fishbein, in press). For example, one meta-analysis found that the correlation between object-based attitude and behavior averaged only 0.13, whereas the correlation between behavioral attitude and the behavior itself averaged 0.54 (Kraus 1995). Thus, better understanding the theoretical relationships within the user satisfaction literature can help bridge such equivocal findings while offering system designers a way to influence usage through design based on system and information characteristics.

2.3. An Integrated Model of User Satisfaction and Technology Acceptance

The investigation of relationships among object-based beliefs, attitudes, and behaviors has been an ongoing challenge in the attitude-behavior literature:

> If there is one clear conclusion to be derived from the work on the attitude-behavior relation it is that general attitudes will usually not provide a good basis for predicting and explaining single behaviors with respect to the attitude object; correlations of single behaviors with general attitudes tend to be modest at best (Ajzen and Fishbein, in press, p. 28).

For accurate prediction, beliefs and attitudes must be specified in a manner that is consistent in time, target, and context with the behavior of interest (Fishbein and Ajzen 1975). This is often referred to as the *correspondence principle* (Fishbein and Ajzen 1975) and is at the core of the power of models such as TAM where beliefs and attitudes about a specific behavior (e.g., the use of an e-mail system), in a particular context (e.g., work), at a particular point in time (e.g., over the next month) are found to be predictive of intention and behavior. Given this, we begin to construct our research model with the right half of Figure 3. Fully

Figure 2 The User Satisfaction Research Stream Approach

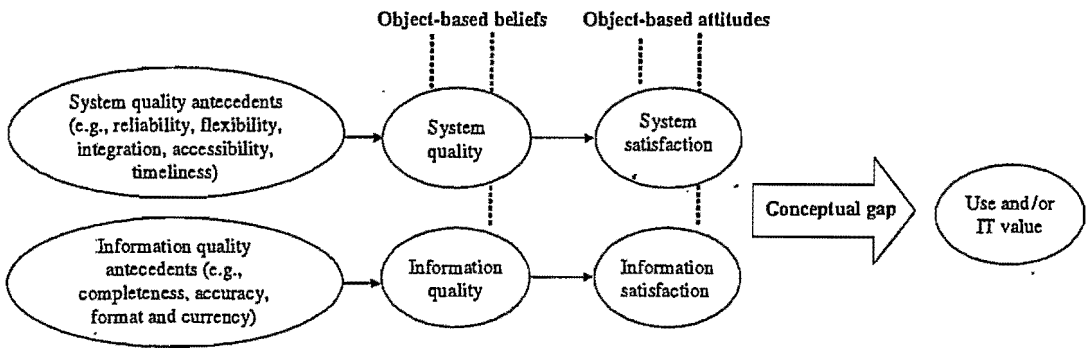
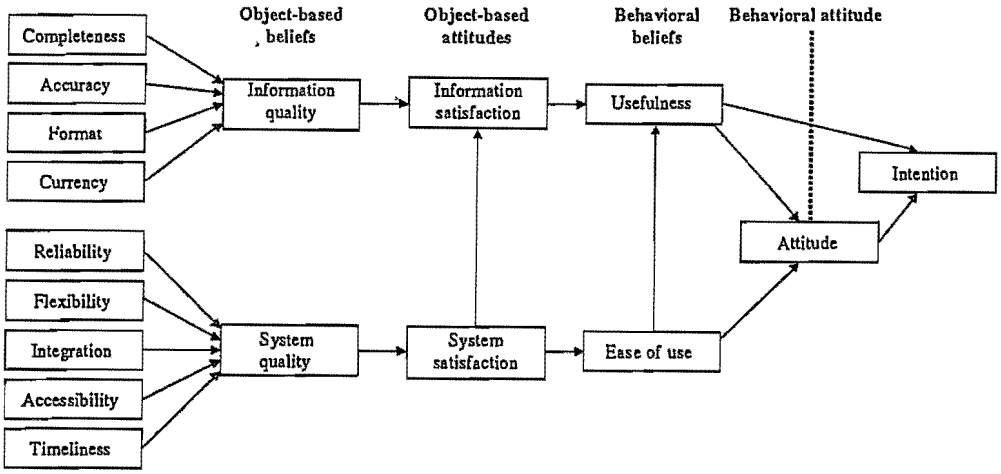


Figure 3 The Proposed Integrated Research Model



consistent with TRA, TAM, and more recent derivations, such as the unified theory of acceptance and use of technology (UTAUT) (Venkatesh et al. 2003), the model proposes that IT usage (the target behavior of interest) is driven by behavioral intention, intention is determined by attitude toward use and usefulness, and usefulness is a function of ease of use. Usefulness and ease of use are both assessments of the consequences of using a system to accomplish some task.

More general object-based attitudes (e.g., attitudes about a system) also can be predictive of behavioral dispositions by influencing the way in which information about the behavior is perceived and judged (Fazio and Olson 2003, Eagly and Chaiken 1993). Theoretically, these serve as external variables that may determine satisfaction with an object, and that level of satisfaction subsequently may influence beliefs about the consequences of using the object (Ajzen and Fishbein, in press). More specifically, Ajzen and Fishbein (1980, p. 9) note that "external variables may influence the beliefs a person holds or the relative importance he attaches to attitudinal and normative considerations."

Ajzen and Fishbein (1980) illustrate the use of object attitudes as external variables using the consumer context. A consumer often forms an attitude toward a particular brand. That attitude is shaped by beliefs about the brand. The consumer may also develop an attitude toward purchase of the brand, which will be influenced by beliefs about the consequences of purchasing the brand. Those beliefs are shaped, at least in part, by the attitude toward the brand itself. In the context of IT, beliefs about using the system to accomplish a particular task will be shaped, in part, by the attitude toward the system itself; indirectly these beliefs will shape attitude toward use and the eventual usage behavior.

Given this, we introduce the left side of Figure 3, which represents the user satisfaction literature. The far-left side of the model specifies key antecedents to information and system quality. These specific factors are derived from a decomposition and integration of factors identified in the user satisfaction literature (see Table 1). Although we believe these dimensions have general applicability, it may be that the relative importance of each is contingent on a specific system and setting. For system quality, *reliability* refers to the dependability of system operation, *flexibility* refers to the way the system adapts to changing demands of the user, *integration* refers to the way the system allows data to be integrated from various sources, *accessibility* refers to the ease with which information can be accessed or extracted from the system, and *timeliness* refers to the degree to which the system offers timely responses to requests for information or action.[2] It is important to note that each of these

[2] These five antecedents to system quality were selected based on their widespread use, representativeness, and relevance to the IT

factors reflects perceptions of the system itself and the way it delivers information.

Information quality is shaped by four dimensions: *completeness* represents the degree to which the system provides all necessary information; *accuracy* represents the user's perception that the information is correct; *format* represents the user's perception of how well the information is presented; and *currency* represents the user's perception of the degree to which the information is up to date.[3] These dimensions determine the user's perception of the quality of the information included in the system.

Next, we assert that information and system quality beliefs shape attitudes about information and system satisfaction, respectively.[4] This is supported by the concept from the attitude behavior literature that beliefs about objects (in this case, system and information quality) are linked to attitude toward an object (in this case, system and information satisfaction) (Ajzen and Fishbein 1980).

At this point, information and system satisfaction represent object-based attitudes that serve as external variables shaping behavioral beliefs. Satisfaction with the information produced by the system will influence perceptions of usefulness. That is, the higher the overall satisfaction with the information, the more likely one will find the application of that information useful in enhancing work performance. A similar effect is anticipated in terms of system satisfaction. System satisfaction represents a degree of favorableness with respect to the system and the mechanics of interaction. The more satisfied one is with the system itself, the more likely one is to find the system to be easy to use.

Consistent with the notion that ease of use will influence perceptions of usefulness, our model hypothesizes that system satisfaction will influence information satisfaction. Being able to effectively interact with the system is a necessary condition to obtaining useful information from it. Thus, an individual's level of satisfaction with the system is likely to influence his or her sense of satisfaction with the information it produces.

To summarize, our models suggest that the technology acceptance literature and the parallel user satisfaction stream are not competing approaches to understanding IT usage and value. Rather, they represent complementary steps in a causal chain from key characteristics of system design, to beliefs and expectations about outcomes that ultimately determine usage. Next, we present a preliminary empirical test of the proposed model to assess the aptness of the proposed relationships. The test is based on a sample of 465 users of data warehousing predefined reporting software from 7 different organizations.

3. Method

3.1. Instrument Development

The development of the survey instrument was patterned after the process proposed by Moore and Benbasat (1991). First, groups of questions were compiled from validated instruments to represent each construct, and wording was modified to fit the data warehousing context to be studied. Next, 10 professors and graduate students sorted the 88 initial items into 17 separate categories, identifying ambiguous or poorly worded items. Items were removed, and minor wording changes were made prior to a second round of sorting, which did not uncover further problems. The three items that were categorized most accurately were selected for each construct and included in a random order on the survey instrument.[5] Each question was measured on a 7-point, Likert-type scale, ranging from 1 (strongly disagree) to 7 (strongly agree).

context that will be explored in this study. This list is not necessarily exhaustive.

[3] These four antecedents to information quality were selected based on their widespread use, representativeness, and relevance to the IT context that will be explored in this study. This list is not necessarily exhaustive.

[4] User satisfaction instruments also refer to other categories of object beliefs, such as service quality that could be included in this model. However, consistent with Seddon (1997), when the focus of the model is on the use of an application, we treat only the system and information characteristics, rather than the broader set of factors that might be used to evaluate satisfaction with overall IT services. This is not to say that such factors are not important, but rather that they are focused on the broader target of the IS function rather than on the individual application.

[5] Only two questions were included for information satisfaction and system satisfaction to reduce redundancy.

The context of the survey instrument was the success of data warehousing predefined reporting software. Predefined reporting software was installed and managed by the data warehousing project team and run by users on a regular basis to provide predetermined information. This context was chosen because of its importance and widespread use in practice. It was hoped that widespread interest in the topic of data warehousing would encourage individual and corporate participation in the study.

Before implementing the survey, the instrument was reviewed by academics and practitioners with knowledge of survey design, IS success, and data warehousing. Minor changes were made based on their suggestions. The resulting survey was then pilot tested using respondents from a large public university to identify problems with the instruments' wording, content, format, and procedures. For this pilot test, surveys were distributed to 250 active users of the university's data warehouse; 73 responded, resulting in a 29% response rate. Pilot participants completed the instruments and provided written comments about length, wording, and instructions. Two of the participants were interviewed to gain a richer understanding of the feedback. Each construct in the pilot test showed internal consistency levels exceeding 0.70, as measured by Cronbach's alpha (Nunnally 1978).

Based on the results of the pilot sample, minor modifications were made to the survey design. The final survey included 76 items representing the 17 constructs identified in Figure 3, as well as a series of demographic and self-reported usage items. The specified items, organized by construct, are shown in Table 2.

3.2. Sample

To obtain study participants, an e-mail announcement was sent to members of The Data Warehousing Institute, offering a free study to assess the success of their organization's data warehousing data access software. Seven organizations from a variety of industries (e.g., health care, consumer goods, financial services, and government) agreed to participate. Each organization was asked to distribute paper-based surveys to all of the active users of its data warehouse. All surveys were confidential; no identifying personal information was required. At each organization, the study contact collected the completed surveys and returned them to the researchers. Response rates varied across organizations (see Table 3), with an overall study response rate of 21%, yielding 465 completed surveys.

The average age of the respondents was 42 years, and 40% were male. The respondents had an average of 12 years tenure with their organization and 18 years average total work experience. Their positions in the organizations varied from clerical to senior management—58% were analysts; they represented different functional areas across the organization. The demographic profile of the sample is shown in Table 4.

The respondents were direct, voluntary users of data warehousing predefined reporting software. On the survey, they identified their absolute usage of the system and their use relative to opportunity. Both absolute and relative usage were measured using a 1 to 7 Likert-type scale, with 1 representing low use and 7 representing high use. The averages for absolute usage and relative usage were 3.6 and 4, respectively, suggesting that the respondents, on average, had a reasonable level of experience using the data warehouse software. The standard deviations for absolute (1.95) and relative usage (1.46) also suggest that there was reasonable variance across the sample in usage experience. All users accessed warehouses that had been in place for at least six months.

4. Results

The research model was tested using partial least squares (PLS), a structural modeling technique that is well suited for highly complex predictive models (Barclay et al. 1995, Chin 1998, Lohmoller 1989, Wold and Joreskog 1982). PLS was most appropriate given the large number of constructs that resulted when the satisfaction and usage models were combined. PLS Graph version 2.91 (Chin and Frye 1996) was used for the analysis, and the bootstrap resampling method (100 resamples) was used to determine the significance of the paths within the structural model.

4.1. Measurement Model

The test of the measurement model includes the estimation of internal consistency and the convergent and discriminant validity of the instrument

Table 2　Survey Items and Measurement Properties

Construct and item	Mean	St. dev.
Completeness		
$\alpha = 0.90$		
Fornell = 0.94		
—— provides me with a complete set of information.	4.58	1.77
—— produces comprehensive information.	4.88	1.70
—— provides me with all the information I need.	4.15	1.87
Format		
$\alpha = 0.89$		
Fornell = 0.92		
The information provided by —— is well formatted.	4.93	1.68
The information provided by —— is well laid out.	5.10	1.57
The information provided by —— is clearly presented on the screen.	5.23	1.55
Accuracy		
$\alpha = 0.87$		
Fornell = 0.90		
—— produces correct information.	5.14	1.60
There are few errors in the information I obtain from ——.	4.75	1.78
The information provided by —— is accurate.	5.04	1.66
Currency		
$\alpha = 0.93$		
Fornell = 0.94		
—— provides me with the most recent information.	5.05	1.87
—— produces the most current information.	4.96	1.79
The information from —— is always up to date.	4.71	1.77
Information quality		
$\alpha = 0.94$		
Fornell = 0.94		
Overall, I would give the information from —— high marks.	5.09	1.68
Overall, I would give the information provided by —— a high rating in terms of quality.	5.10	1.63
In general, —— provides me with high-quality information.	5.11	1.61
Reliability		
$\alpha = 0.90$		
Fornell = 0.93		
—— operates reliably.	5.10	1.73
—— performs reliably.	5.15	1.66
The operation of —— is dependable.	5.10	1.56
Accessibility		
$\alpha = 0.90$		
Fornell = 0.92		
—— allows information to be readily accessible to me.	5.27	1.70
—— makes information very accessible.	5.16	1.69
—— makes information easy to access.	5.14	1.70
Flexibility		
$\alpha = 0.86$		
Fornell = 0.90		
—— can be adapted to meet a variety of needs.	4.28	1.99
—— can flexibly adjust to new demands or conditions.	3.73	1.86
—— is versatile in addressing needs as they arise.	4.00	1.83
Integration		
$\alpha = 0.89$		
Fornell = 0.91		
—— effectively integrates data from different areas of the company.	4.78	1.89

Table 2 (cont'd.)

Construct and item	Mean	St. dev.
—— pulls together information that used to come from different places in the company.	5.14	1.77
—— effectively combines data from different areas of the company.	4.93	1.77
Timeliness		
$\alpha = 0.80$		
Fornell = 0.87		
It takes too long for —— to respond to my requests. (RC)	4.26	1.90
—— provides information in a timely fashion.	5.07	1.67
—— returns answers to my requests quickly.	4.90	1.72
System quality		
$\alpha = 0.91$		
Fornell = 0.94		
In terms of system quality, I would rate —— highly.	4.91	1.69
Overall, —— is of high quality.	5.12	1.55
Overall, I would give the quality of —— a high rating.	4.97	1.62
Information satisfaction		
$\alpha = 0.93$		
Fornell = 0.96		
Overall, the information I get from —— is very satisfying.	4.89	1.80
I am very satisfied with the information I receive from ——.	4.84	1.78
System satisfaction		
$\alpha = 0.92$		
Fornell = 0.95		
All things considered, I am very satisfied with ——.	4.61	1.94
Overall, my interaction with —— is very satisfying.	4.65	1.82
Attitude		
$\alpha = 0.89$		
Fornell = 0.91		
Using —— is (not enjoyable/ very enjoyable).	4.13	1.86
Overall, using —— is a (unpleasant/pleasant) experience.	4.89	1.79
My attitude toward using —— is (very unfavorable/very favorable).	4.98	1.77
Intention		
$\alpha = 0.87$		
Fornell = 0.92		
I intend to use —— as a routine part of my job over the next year.	5.13	1.94
I intend to use —— at every opportunity over the next year.	4.80	1.93
I plan to increase my use of —— over the next year.	4.64	1.88
Ease of use		
$\alpha = 0.85$		
Fornell = 0.89		
—— is easy to use.	5.31	1.82
It is easy to get —— to do what I want it to do.	4.39	1.98
—— is easy to operate.	5.20	1.84
Usefulness		
$\alpha = 0.82$		
Fornell = 0.88		
Using —— improves my ability to make good decisions.	5.04	1.65
—— allows me to get my work done more quickly.	4.84	1.80
Using —— enhances my effectiveness on the job.	5.04	1.76

Note. RC = reverse coded.

Table 3 Survey Response Rates

Company	Surveys sent*	Surveys returned	Response rate (percent)
A-Health care	129	40	31
B-Packaged goods	300	92	31
C-Financial services	179	23	13
D-Health care	108	42	39
E-Public sector	1,200	172	14
F-Public sector	231	61	26
G-Public sector	66	35	53
Overall	2,213	465	21

*Note that this represents the number of surveys sent to each company. We cannot be certain that all surveys sent were distributed to data warehouse users. Thus, our effective response rate is likely somewhat higher that reported here.

Table 4 Demographic Profile of Respondents

	Number	Percent
Organizational level:		
Senior management	13	3
Middle management	95	22
First-level supervisor	48	11
Analyst	257	58
Clerical	27	6
Functional area:		
Accounting	22	5
Finance	79	17
Human resources	22	5
Information systems	37	8
Marketing and sales	82	18
Research and development	96	21
Other	116	25
Gender:		
Male	180	40
Female	270	60
Average age:		42 years
Average years at company:		12 years
Average years in workforce:		18 years

items. Table 2 lists the survey scales and their internal consistency reliabilities. All reliability measures were 0.8 or higher, well above the recommended level of 0.70, indicating adequate internal consistency (Nunnally 1978).

Although some of the variable intercorrelations were quite high (ranging from 0.36 to 0.85), the items demonstrated satisfactory convergent and discriminant validity. Convergent validity is adequate when constructs have an average variance extracted (AVE) of at least 0.5 (Fornell and Larcker 1981). For satisfactory discriminant validity, the AVE from the construct should be greater than the variance shared between the construct and other constructs in the model (Chin 1998). Table 5 lists the correlation matrix, with correlations among constructs and the square root of AVE on the diagonal. In all cases, the AVE for each construct is larger than the correlation of that construct with all other constructs in the model.

Table 5 Correlations of Latent Variables†

	COMP	ACCU	FORM	CURR	RELI	FLEX	INTE	TIME	ACCE	INFQ	SYSQ	INTN	ATTI	EASE	USEF	SYSS	INFS
Completeness	**0.91**																
Accuracy	0.55*	**0.87**															
Format	0.66	0.49	**0.89**														
Currency	0.57	0.61	0.48	**0.92**													
Reliability	0.62	0.68	0.57	0.59	**0.90**												
Flexibility	0.68	0.33	0.47	0.39	0.41	**0.86**											
Integration	0.69	0.46	0.50	0.50	0.50	0.48	**0.88**										
Timeliness	0.55	0.54	0.52	0.54	0.74	0.46	0.47	**0.83**									
Accessibility	0.70	0.54	0.63	0.55	0.70	0.54	0.61	0.70	**0.89**								
Info. quality	0.74	0.76	0.64	0.64	0.73	0.54	0.63	0.60	0.71	**0.91**							
System quality	0.77	0.71	0.71	0.56	0.77	0.57	0.57	0.67	0.77	0.85	**0.91**						
Intention	0.53	0.36	0.40	0.39	0.41	0.51	0.42	0.43	0.58	0.53	0.57	**0.89**					
Attitude	0.59	0.49	0.63	0.44	0.61	0.56	0.47	0.62	0.70	0.67	0.75	0.71	**0.87**				
Ease of use	0.52	0.42	0.57	0.38	0.58	0.44	0.45	0.56	0.68	0.58	0.69	0.55	0.77	**0.85**			
Usefulness	0.60	0.47	0.49	0.47	0.54	0.54	0.53	0.52	0.66	0.69	0.67	0.76	0.75	0.65	**0.85**		
System sat.	0.66	0.41	0.60	0.39	0.60	0.58	0.51	0.57	0.71	0.67	0.75	0.67	0.84	0.81	0.77	**0.95**	
Information sat.	0.67	0.52	0.55	0.51	0.61	0.58	0.54	0.54	0.63	0.77	0.72	0.66	0.73	0.61	0.80	0.79	**0.96**

*All correlations are significant at the 0.001 level.
†Diagonal elements are the square root of AVE. These values should exceed the interconstruct correlations for adequate discriminant validity. This condition is satisfied for each construct.

Discriminant and convergent validity are further confirmed when individual items load above 0.50 on their associated factors and when the loadings within construct are higher than those across constructs. The appendix contains the loadings and cross-loadings for items used in this study; all items loaded on their constructs as expected. Furthermore, all items loaded more highly on their construct than they loaded on any other construct, and in all but one case among the 784 cross loadings the differences were greater than 0.10.

Finally, the data were tested for multicollinearity. We tested for all potential collinearity problems that had more than one predictor construct. In all cases, the variance inflation factor was below the 5.0 level.

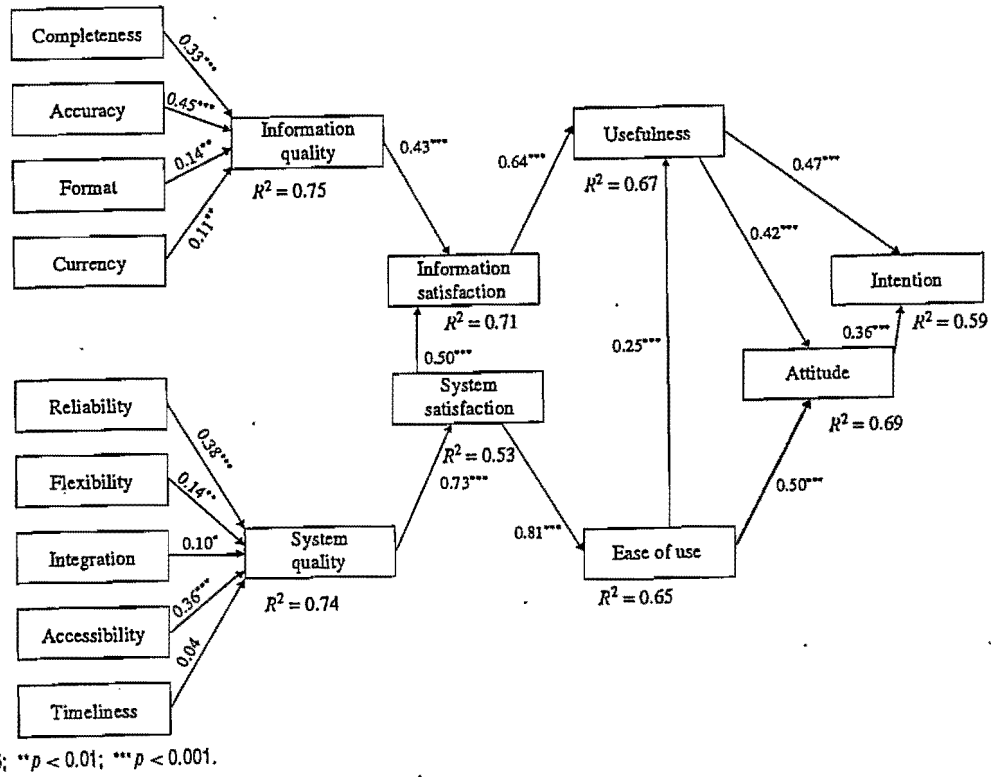
4.2. Structural Model

The test of the structural model includes estimates of the path coefficients, which indicate the strengths of the relationships between the dependent and independent variables, and the R^2 values, which represent the amount of variance explained by the independent variables. Together, the R^2 and the path coefficients (loadings and significance) indicate how well the data support the hypothesized model.

Figure 4 shows the results of the test of the hypothesized structural model. The paths specified in TAM are all significant with the direct and indirect effects of usefulness, ease of use, and attitude toward use accounting for 59% of the variance in intention. As predicted, information satisfaction (0.64) had a significant influence on perceived usefulness and accounted for 67% of the variance in perceived usefulness. System satisfaction (0.81) had a significant influence on perceived ease of use and accounted for 65% of the variance in perceived ease of use.

As expected, information quality (0.43) and system satisfaction (0.50) had significant influences on information satisfaction, accounting for 71% of the variance in that measure. System quality also was a significant determinant of system satisfaction (0.73), accounting for 53% of its variance. Completeness (0.33), accuracy (0.45), format (0.14), and currency (0.11) were all significantly related to information quality and collectively account for 75% of the variance in information quality. Reliability (0.38), flexibility (0.14), integration (0.10), and accessibility (0.36) were all significant determinants of system quality, whereas timeliness was not. The first three factors together accounted for 74% of the variance in system quality.

Figure 4 Research Model Results



*$p < 0.05$; **$p < 0.01$; ***$p < 0.001$.

國立雲林科技大學
100學年度博士班招生考試試題

系所別：資訊管理系
科　目：資訊科技文獻

請以中文回答下列問題：

第一部份：

Title: Above the Clouds: A Berkeley View of Cloud Computing

1. 試簡略說明本篇論文各章節之主要重點結論為何? (15%)
2. 根據本篇論文 Utility Computing 之商業模式(business model)有那些? 試提出一種本篇論文尚未提及之可行性商業模式。(20%)
3. 根據本篇論文，您提出的商業模式會面臨那些障礙? 您需要考量那些風險? (15%)

第二部份：

Title: Great Principles of Computing

4. 請說明本篇論文的主要論點。(15%)
5. 以作者的觀點，為什麼在 IT 領域的學生在很早期的時候，會輕易的放棄在 IT 領域的鑽研。(15%)
6. 以作者的觀點，他是否認同「可以將 "computing mechanics" 視為 "the science of computing"，而將"design principles"視為 "the art of computing"」，若是，作者所主張的理由是什麼? 若不是，其所反對的理由又是什麼? (20%)

國立雲林科技大學
100學年度博士班招生考試試題

系所別：資訊管理系
科　目：資訊科技文獻

Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz,
Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia
(Comments should be addressed to abovetheclouds@cs.berkeley.edu)

UC Berkeley Reliable Adaptive Distributed Systems Laboratory *
http://radlab.cs.berkeley.edu/

February 10, 2009

KEYWORDS: Cloud Computing, Utility Computing, Internet Datacenters, Distributed System Economics

1 Executive Summary

Cloud Computing, the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. Developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it. They need not be concerned about over-provisioning for a service whose popularity does not meet their predictions, thus wasting costly resources, or under-provisioning for one that becomes wildly popular, thus missing potential customers and revenue. Moreover, companies with large batch-oriented tasks can get results as quickly as their programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours. This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT.

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as *Software as a Service (SaaS)*. The datacenter hardware and software is what we will call a *Cloud*. When a Cloud is made available in a pay-as-you-go manner to the general public, we call it a *Public Cloud*; the service being sold is *Utility Computing*. We use the term *Private Cloud* to refer to internal datacenters of a business or other organization, not made available to the general public. Thus, Cloud Computing is the sum of SaaS and Utility Computing, but does not include Private Clouds. People can be users or providers of SaaS, or users or providers of Utility Computing. We focus on SaaS Providers (Cloud Users) and Cloud Providers, which have received less attention than SaaS Users.

From a hardware point of view, three aspects are new in Cloud Computing.

1. *The illusion of infinite computing resources available on demand*, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning.
2. *The elimination of an up-front commitment by Cloud users*, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.
3. *The ability to pay for use of computing resources on a short-term basis as needed* (e.g., processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

We argue that the construction and operation of extremely large-scale, commodity-computer datacenters at low-cost locations was the key necessary enabler of Cloud Computing, for they uncovered the factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies

*The RAD Lab's existence is due to the generous support of the founding members Google, Microsoft, and Sun Microsystems and of the affiliate members Amazon Web Services, Cisco Systems, Facebook, Hewlett-Packard, IBM, NEC, Network Appliance, Oracle, Siemens, and VMware; by matching funds from the State of California's MICRO program (grants 06-152, 07-010, 06-148, 07-012, 06-146, 07-009, 06-147, 07-013, 06-149, 06-150, and 07-008) and the University of California Industry/University Cooperative Research Program (UC Discovery) grant COM07-10240; and by the National Science Foundation (grant #CNS-0509559).

of scale. These factors, combined with statistical multiplexing to increase utilization compared a private cloud, meant that cloud computing could offer services below the costs of a medium-sized datacenter and yet still make a good profit.

Any application needs a model of computation, a model of storage, and a model of communication. The statistical multiplexing necessary to achieve elasticity and the illusion of infinite capacity requires each of these resources to be virtualized to hide the implementation of how they are multiplexed and shared. Our view is that different utility computing offerings will be distinguished based on the level of abstraction presented to the programmer and the level of management of the resources.

Amazon EC2 is at one end of the spectrum. An EC2 instance looks much like physical hardware, and users can control nearly the entire software stack, from the kernel upwards. This low level makes it inherently difficult for Amazon to offer automatic scalability and failover, because the semantics associated with replication and other state management issues are highly application-dependent. At the other extreme of the spectrum are application domain-specific platforms such as Google AppEngine. AppEngine is targeted exclusively at traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. AppEngine's impressive automatic scaling and high-availability mechanisms, and the proprietary MegaStore data storage available to AppEngine applications, all rely on these constraints. Applications for Microsoft's Azure are written using the .NET libraries, and compiled to the Common Language Runtime, a language-independent managed environment. Thus, Azure is intermediate between application frameworks like AppEngine and hardware virtual machines like EC2.

When is Utility Computing preferable to running a Private Cloud? A first case is when demand for a service varies with time. Provisioning a data center for the peak load it must sustain a few days per month leads to underutilization at other times, for example. Instead, Cloud Computing lets an organization pay by the hour for computing resources, potentially leading to cost savings even if the hourly rate to rent a machine from a cloud provider is higher than the rate to own one. A second case is when demand is unknown in advance. For example, a web startup will need to support a spike in demand when it becomes popular, followed potentially by a reduction once some of the visitors turn away. Finally, organizations that perform batch analytics can use the "cost associativity" of cloud computing to finish computations faster: using 1000 EC2 machines for 1 hour costs the same as using 1 machine for 1000 hours. For the first case of a web business with varying demand over time and revenue proportional to user hours, we have captured the tradeoff in the equation below.

$$\text{UserHours}_{\text{cloud}} \times (\text{revenue} - \text{Cost}_{\text{cloud}}) \geq \text{UserHours}_{\text{datacenter}} \times \left(\text{revenue} - \frac{\text{Cost}_{\text{datacenter}}}{\text{Utilization}}\right) \quad (1)$$

The left-hand side multiplies the net revenue per user-hour by the number of user-hours, giving the expected profit from using Cloud Computing. The right-hand side performs the same calculation for a fixed-capacity datacenter by factoring in the average utilization, including nonpeak workloads, of the datacenter. Whichever side is greater represents the opportunity for higher profit.

Table 1 below previews our ranked list of critical obstacles to growth of Cloud Computing in Section 7. The first three concern *adoption*, the next five affect *growth*, and the last two are *policy and business* obstacles. Each obstacle is paired with an *opportunity*, ranging from product development to research projects, which can overcome that obstacle.

We predict Cloud Computing *will* grow, so developers should take it into account. All levels should aim at horizontal scalability of virtual machines over the efficiency on a single VM. In addition

1. *Applications Software* needs to both scale *down* rapidly as well as scale up, which is a new requirement. Such software also needs a pay-for-use licensing model to match needs of Cloud Computing.

2. *Infrastructure Software* needs to be aware that it is no longer running on bare metal but on VMs. Moreover, it needs to have billing built in from the beginning.

3. *Hardware Systems* should be designed at the scale of a container (at least a dozen racks), which will be is the minimum purchase size. Cost of operation will match performance and cost of purchase in importance, rewarding *energy proportionality* such as by putting idle portions of the memory, disk, and network into low power mode. Processors should work well with VMs, flash memory should be added to the memory hierarchy, and LAN switches and WAN routers must improve in bandwidth and cost.

2 Cloud Computing: An Old Idea Whose Time Has (Finally) Come

Cloud Computing is a new term for a long-held dream of computing as a utility [35], which has recently emerged as a commercial reality. Cloud Computing is likely to have the same impact on software that foundries have had on the

Table 1: Quick Preview of Top 10 Obstacles to and Opportunities for Growth of Cloud Computing.

	Obstacle	Opportunity
1	Availability of Service	Use Multiple Cloud Providers; Use Elasticity to Prevent DDOS
2	Data Lock-In	Standardize APIs; Compatible SW to enable Surge Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls; Geographical Data Storage
4	Data Transfer Bottlenecks	FedExing Disks; Data Backup/Archival; Higher BW Switches
5	Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on ML; Snapshots for Conservation
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses; Bulk use sales

hardware industry. At one time, leading hardware companies required a captive semiconductor fabrication facility, and companies had to be large enough to afford to build and operate it economically. However, processing equipment doubled in price every technology generation. A semiconductor fabrication line costs over $3B today, so only a handful of major "merchant" companies with very high chip volumes, such as Intel and Samsung, can still justify owning and operating their own fabrication lines. This motivated the rise of semiconductor foundries that build chips for others, such as Taiwan Semiconductor Manufacturing Company (TSMC). Foundries enable "fab-less" semiconductor chip companies whose value is in innovative chip design: A company such as nVidia can now be successful in the chip business without the capital, operational expenses, and risks associated with owning a state-of-the-art fabrication line. Conversely, companies with fabrication lines can time-multiplex their use among the products of many fab-less companies, to lower the risk of not having enough successful products to amortize operational costs. Similarly, the advantages of the economy of scale and statistical multiplexing may ultimately lead to a handful of Cloud Computing providers who can amortize the cost of their large datacenters over the products of many "datacenter-less" companies.

Cloud Computing has been talked about [10], blogged about [13, 25], written about [15, 37, 38] and been featured in the title of workshops, conferences, and even magazines. Nevertheless, confusion remains about exactly what it is and when it's useful, causing Oracle's CEO to vent his frustration:

> *The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything that we already do.... I don't understand what we would do differently in the light of Cloud Computing other than change the wording of some of our ads.*
>
> Larry Ellison, quoted in the *Wall Street Journal*, September 26, 2008

These remarks are echoed more mildly by Hewlett-Packard's Vice President of European Software Sales:

> *A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of "the cloud."*
>
> Andy Isherwood, quoted in *ZDnet News*, December 11, 2008

Richard Stallman, known for his advocacy of "free software", thinks Cloud Computing is a trap for users—if applications and data are managed "in the cloud", users might become dependent on proprietary systems whose costs will escalate or whose terms of service might be changed unilaterally and adversely:

> *It's stupidity. It's worse than stupidity: it's a marketing hype campaign. Somebody is saying this is inevitable — and whenever you hear somebody saying that, it's very likely to be a set of businesses campaigning to make it true.*
>
> Richard Stallman, quoted in *The Guardian*, September 29, 2008

Our goal in this paper to clarify terms, provide simple formulas to quantify comparisons between of cloud and conventional Computing, and identify the top technical and non-technical obstacles and opportunities of Cloud Computing. Our view is shaped in part by working since 2005 in the UC Berkeley RAD Lab and in part as users of Amazon Web Services since January 2008 in conducting our research and our teaching. The RAD Lab's research agenda is to invent technology that leverages machine learning to help automate the operation of datacenters for scalable Internet services. We spent six months brainstorming about Cloud Computing, leading to this paper that tries to answer the following questions:

- What is Cloud Computing, and how is it different from previous paradigm shifts such as Software as a Service (SaaS)?
- Why is Cloud Computing poised to take off now, whereas previous attempts have foundered?
- What does it take to become a Cloud Computing provider, and why would a company consider becoming one?
- What new opportunities are either enabled by or potential drivers of Cloud Computing?
- How might we classify current Cloud Computing offerings across a spectrum, and how do the technical and business challenges differ depending on where in the spectrum a particular offering lies?
- What, if any, are the new economic models enabled by Cloud Computing, and how can a service operator decide whether to move to the cloud or stay in a private datacenter?
- What are the top 10 obstacles to the success of Cloud Computing—and the corresponding top 10 opportunities available for overcoming the obstacles?
- What changes should be made to the design of future applications software, infrastructure software, and hardware to match the needs and opportunities of Cloud Computing?

3 What is Cloud Computing?

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as *Software as a Service (SaaS)*, so we use that term. The datacenter hardware and software is what we will call a *Cloud.*

When a Cloud is made available in a pay-as-you-go manner to the public, we call it a *Public Cloud*; the service being sold is *Utility Computing*. Current examples of public Utility Computing include Amazon Web Services, Google AppEngine, and Microsoft Azure. We use the term *Private Cloud* to refer to internal datacenters of a business or other organization that are not made available to the public. Thus, Cloud Computing is the sum of SaaS and Utility Computing, but does not normally include Private Clouds. We'll generally use Cloud Computing, replacing it with one of the other terms only when clarity demands it. Figure 1 shows the roles of the people as users or providers of these layers of Cloud Computing, and we'll use those terms to help make our arguments clear.

The advantages of SaaS to both end users and service providers are well understood. Service providers enjoy greatly simplified software installation and maintenance and centralized control over versioning; end users can access the service "anytime, anywhere", share data and collaborate more easily, and keep their data stored safely in the infrastructure. Cloud Computing does not change these arguments, but it does give more application providers the choice of deploying their product as SaaS without provisioning a datacenter: just as the emergence of semiconductor foundries gave chip companies the opportunity to design and sell chips without owning a fab, Cloud Computing allows deploying SaaS—and scaling on demand—without building or provisioning a datacenter. Analogously to how SaaS allows the user to offload some problems to the SaaS provider, the SaaS provider can now offload some of *his* problems to the Cloud Computing provider. From now on, we will focus on issues related to the potential SaaS Provider (Cloud User) and to the Cloud Providers, which have received less attention.

We will eschew terminology such as "X as a service (XaaS)"; values of *X* we have seen in print include Infrastructure, Hardware, and Platform, but we were unable to agree even among ourselves what the precise differences among them might be.[1] *(We are using Endnotes instead of footnotes. Go to page 20 at the end of paper to read the notes, which have more details.)* Instead, we present a simple classification of Utility Computing services in Section 5 that focuses on the tradeoffs among programmer convenience, flexibility, and portability, from both the cloud provider's and the cloud user's point of view.

From a hardware point of view, three aspects are new in Cloud Computing [42]:

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning;
2. The elimination of an up-front commitment by Cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs; and
3. The ability to pay for use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

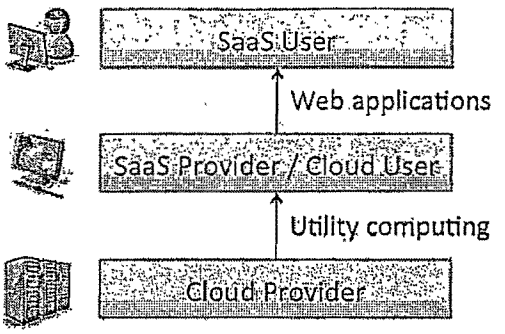


Figure 1: Users and Providers of Cloud Computing. The benefits of SaaS to both SaaS users and SaaS providers are well documented, so we focus on Cloud Computing's effects on Cloud Providers and SaaS Providers/Cloud users. The top level can be recursive, in that SaaS providers can also be a SaaS users. For example, a mashup provider of rental maps might be a user of the Craigslist and Google maps services.

We will argue that all three are important to the technical and economic changes made possible by Cloud Computing. Indeed, past efforts at utility computing failed, and we note that in each case one or two of these three critical characteristics were missing. For example, Intel Computing Services in 2000-2001 required negotiating a contract and longer-term use than per hour.

As a successful example, Elastic Compute Cloud (EC2) from Amazon Web Services (AWS) sells 1.0-GHz x86 ISA "slices" for 10 cents per hour, and a new "slice", or *instance*, can be added in 2 to 5 minutes. Amazon's Scalable Storage Service (S3) charges $0.12 to $0.15 per gigabyte-month, with additional bandwidth charges of $0.10 to $0.15 per gigabyte to move data in to and out of AWS over the Internet. Amazon's bet is that by statistically multiplexing multiple instances onto a single physical box, that box can be simultaneously rented to many customers who will not in general interfere with each others' usage (see Section 7).

While the attraction to Cloud Computing users (SaaS providers) is clear, who would become a Cloud Computing provider, and why? To begin with, realizing the economies of scale afforded by statistical multiplexing and bulk purchasing requires the construction of extremely large datacenters.

Building, provisioning, and launching such a facility is a hundred-million-dollar undertaking. However, because of the phenomenal growth of Web services through the early 2000's, many large Internet companies, including Amazon, eBay, Google, Microsoft and others, were already doing so. Equally important, these companies also had to develop scalable software infrastructure (such as MapReduce, the Google File System, BigTable, and Dynamo [16, 20, 14, 17]) and the operational expertise to armor their datacenters against potential physical and electronic attacks.

Therefore, a necessary but not sufficient condition for a company to become a Cloud Computing provider is that it must have existing investments not only in very large datacenters, but also in large-scale software infrastructure and operational expertise required to run them. Given these conditions, a variety of factors might influence these companies to become Cloud Computing providers:

1. **Make a lot of money.** Although 10 cents per server-hour seems low, Table 2 summarizes James Hamilton's estimates [23] that very large datacenters (tens of thousands of computers) can purchase hardware, network bandwidth, and power for 1/5 to 1/7 the prices offered to a medium-sized (hundreds or thousands of computers) datacenter. Further, the fixed costs of software development and deployment can be amortized over many more machines. Others estimate the price advantage as a factor of 3 to 5 [37, 10]. Thus, a sufficiently large company could leverage these economies of scale to offer a service well below the costs of a medium-sized company and still make a tidy profit.

2. **Leverage existing investment.** Adding Cloud Computing services on top of existing infrastructure provides a new revenue stream at (ideally) low incremental cost, helping to amortize the large investments of datacenters. Indeed, according to Werner Vogels, Amazon's CTO, many Amazon Web Services technologies were initially developed for Amazon's internal operations [42].

3. **Defend a franchise.** As conventional server and enterprise applications embrace Cloud Computing, vendors with an established franchise in those applications would be motivated to provide a cloud option of their own. For example, Microsoft Azure provides an immediate path for migrating existing customers of Microsoft enterprise applications to a cloud environment.

Table 2: Economies of scale in 2006 for medium-sized datacenter (≈1000 servers) vs. very large datacenter (≈50,000 servers). [24]

Technology	Cost in Medium-sized DC	Cost in Very Large DC	Ratio
Network	$95 per Mbit/sec/month	$13 per Mbit/sec/month	7.1
Storage	$2.20 per GByte / month	$0.40 per GByte / month	5.7
Administration	≈140 Servers / Administrator	>1000 Servers / Administrator	7.1

Table 3: Price of kilowatt-hours of electricity by region [7].

Price per KWH	Where	Possible Reasons Why
3.6¢	Idaho	Hydroelectric power; not sent long distance
10.0¢	California	Electricity transmitted long distance over the grid; limited transmission lines in Bay Area; no coal fired electricity allowed in California.
18.0¢	Hawaii	Must ship fuel to generate electricity

4. **Attack an incumbent.** A company with the requisite datacenter and software resources might want to establish a beachhead in this space before a single "800 pound gorilla" emerges. Google AppEngine provides an alternative path to cloud deployment whose appeal lies in its automation of many of the scalability and load balancing features that developers might otherwise have to build for themselves.

5. **Leverage customer relationships.** IT service organizations such as IBM Global Services have extensive customer relationships through their service offerings. Providing a branded Cloud Computing offering gives those customers an anxiety-free migration path that preserves both parties' investments in the customer relationship.

6. **Become a platform.** Facebook's initiative to enable plug-in applications is a great fit for cloud computing, as we will see, and indeed one infrastructure provider for Facebook plug-in applications is Joyent, a cloud provider. Yet Facebook's motivation was to make their social-networking application a new development platform.

Several Cloud Computing (and conventional computing) datacenters are being built in seemingly surprising locations, such as Quincy, Washington (Google, Microsoft, Yahoo!, and others) and San Antonio, Texas (Microsoft, US National Security Agency, others). The motivation behind choosing these locales is that the costs for electricity, cooling, labor, property purchase costs, and taxes are geographically variable, and of these costs, electricity and cooling alone can account for a third of the costs of the datacenter. Table 3 shows the cost of electricity in different locales [10]. Physics tells us it's easier to ship photons than electrons; that is, it's cheaper to ship data over fiber optic cables than to ship electricity over high-voltage transmission lines.

4 Clouds in a Perfect Storm: Why Now, Not Then?

Although we argue that the construction and operation of extremely large scale commodity-computer datacenters was the key necessary enabler of Cloud Computing, additional technology trends and new business models also played a key role in making it a reality this time around. Once Cloud Computing was "off the ground," new application opportunities and usage models were discovered that would not have made sense previously.

4.1 New Technology Trends and Business Models

Accompanying the emergence of Web 2.0 was a shift from "high-touch, high-margin, high-commitment" provisioning of service "low-touch, low-margin, low-commitment" self-service. For example, in Web 1.0, accepting credit card payments from strangers required a contractual arrangement with a payment processing service such as VeriSign or Authorize.net; the arrangement was part of a larger business relationship, making it onerous for an individual or a very small business to accept credit cards online. With the emergence of PayPal, however, any individual can accept credit card payments with no contract, no long-term commitment, and only modest pay-as-you-go transaction fees. The level of "touch" (customer support and relationship management) provided by these services is minimal to nonexistent, but

the fact that the services are now within reach of individuals seems to make this less important. Similarly, individuals' Web pages can now use Google AdSense to realize revenue from ads, rather than setting up a relationship with an ad placement company, such DoubleClick (now acquired by Google). Those ads can provide the business model for Wed 2.0 apps as well. Individuals can distribute Web content using Amazon CloudFront rather than establishing a relationship with a content distribution network such as Akamai.

Amazon Web Services capitalized on this insight in 2006 by providing pay-as-you-go computing with no contract: all customers need is a credit card. A second innovation was selling hardware-level virtual machines cycles, allowing customers to choose their own software stack without disrupting each other while sharing the same hardware and thereby lowering costs further.

4.2 New Application Opportunities

While we have yet to see fundamentally new *types* of applications enabled by Cloud Computing, we believe that several important classes of existing applications will become even more compelling with Cloud Computing and contribute further to its momentum. When Jim Gray examined technological trends in 2003 [21], he concluded that economic necessity mandates putting the data near the application, since the cost of wide-area networking has fallen more slowly (and remains relatively higher) than all other IT hardware costs. Although hardware costs have changed since Gray's analysis, his idea of this "breakeven point" has not. Although we defer a more thorough discussion of Cloud Computing economics to Section 6, we use Gray's insight in examining what kinds of applications represent particularly good opportunities and drivers for Cloud Computing.

Mobile interactive applications. Tim O'Reilly believes that *"the future belongs to services that respond in real time to information provided either by their users or by nonhuman sensors."* [38] Such services will be attracted to the cloud not only because they must be highly available, but also because these services generally rely on large data sets that are most conveniently hosted in large datacenters. This is especially the case for services that combine two or more data sources or other services, e.g., mashups. While not all mobile devices enjoy connectivity to the cloud 100% of the time, the challenge of disconnected operation has been addressed successfully in specific application domains, [2] so we do not see this as a significant obstacle to the appeal of mobile applications.

Parallel batch processing. Although thus far we have concentrated on using Cloud Computing for interactive SaaS, Cloud Computing presents a unique opportunity for batch-processing and analytics jobs that analyze terabytes of data and can take hours to finish. If there is enough data parallelism in the application, users can take advantage of the cloud's new "cost associativity": using hundreds of computers for a short time costs the same as using a few computers for a long time. For example, Peter Harkins, a Senior Engineer at The Washington Post, used 200 EC2 instances (1,407 server hours) to convert 17,481 pages of Hillary Clinton's travel documents into a form more friendly to use on the WWW within nine hours after they were released [3]. Programming abstractions such as Google's MapReduce [16] and its open-source counterpart Hadoop [11] allow programmers to express such tasks while hiding the operational complexity of choreographing parallel execution across hundreds of Cloud Computing servers. Indeed, Cloudera [1] is pursuing commercial opportunities in this space. Again, using Gray's insight, the cost/benefit analysis must weigh the cost of moving large datasets into the cloud against the benefit of potential speedup in the data analysis. When we return to economic models later, we speculate that part of Amazon's motivation to host large public datasets for free [8] may be to mitigate the cost side of this analysis and thereby attract users to purchase Cloud Computing cycles near this data.

The rise of analytics. A special case of compute-intensive batch processing is business analytics. While the large database industry was originally dominated by transaction processing, that demand is leveling off. A growing share of computing resources is now spent on understanding customers, supply chains, buying habits, ranking, and so on. Hence, while online transaction volumes will continue to grow slowly, decision support is growing rapidly, shifting the resource balance in database processing from transactions to business analytics.

Extension of compute-intensive desktop applications. The latest versions of the mathematics software packages Matlab and Mathematica are capable of using Cloud Computing to perform expensive evaluations. Other desktop applications might similarly benet from seamless extension into the cloud. Again, a reasonable test is comparing the cost of computing in the Cloud plus the cost of moving data in and out of the Cloud to the time savings from using the Cloud. Symbolic mathematics involves a great deal of computing per unit of data, making it a domain worth investigating. An interesting alternative model might be to keep the data in the cloud and rely on having sufficient bandwidth to enable suitable visualization and a responsive GUI back to the human user. Offline image rendering or 3D animation might be a similar example: given a compact description of the objects in a 3D scene and the characteristics of the lighting sources, rendering the image is an embarrassingly parallel task with a high computation-to-bytes ratio.

"Earthbound" applications. Some applications that would otherwise be good candidates for the cloud's elasticity and parallelism may be thwarted by data movement costs, the fundamental latency limits of getting into and out of the cloud, or both. For example, while the analytics associated with making long-term financial decisions are appropriate

for the Cloud, stock trading that requires microsecond precision is not. Until the cost (and possibly latency) of wide-area data transfer decrease (see Section 7), such applications may be less obvious candidates for the cloud.

5 Classes of Utility Computing

Any application needs a model of computation, a model of storage and, assuming the application is even trivially distributed, a model of communication. The statistical multiplexing necessary to achieve elasticity and the illusion of infinite capacity requires resources to be virtualized, so that the implementation of how they are multiplexed and shared can be hidden from the programmer. Our view is that different utility computing offerings will be distinguished based on the level of abstraction presented to the programmer and the level of management of the resources.

Amazon EC2 is at one end of the spectrum. An EC2 instance looks much like physical hardware, and users can control nearly the entire software stack, from the kernel upwards. The API exposed is "thin": a few dozen API calls to request and configure the virtualized hardware. There is no *a priori* limit on the kinds of applications that can be hosted; the low level of virtualization—raw CPU cycles, block-device storage, IP-level connectivity—allow developers to code whatever they want. On the other hand, this makes it inherently difficult for Amazon to offer automatic scalability and failover, because the semantics associated with replication and other state management issues are highly application-dependent.

AWS does offer a number of higher-level managed services, including several different managed storage services for use in conjunction with EC2, such as SimpleDB. However, these offerings have higher latency and nonstandard API's, and our understanding is that they are not as widely used as other parts of AWS.

At the other extreme of the spectrum are application domain-specific platforms such as Google AppEngine and Force.com, the SalesForce business software development platform. AppEngine is targeted exclusively at traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. Furthermore, AppEngine applications are expected to be request-reply based, and as such they are severely rationed in how much CPU time they can use in servicing a particular request. AppEngine's impressive automatic scaling and high-availability mechanisms, and the proprietary MegaStore (based on BigTable) data storage available to AppEngine applications, all rely on these constraints. Thus, AppEngine is not suitable for general-purpose computing. Similarly, Force.com is designed to support business applications that run against the salesforce.com database, and nothing else.

Microsoft's Azure is an intermediate point on this spectrum of flexibility vs. programmer convenience. Azure applications are written using the .NET libraries, and compiled to the Common Language Runtime, a language-independent managed environment. The system supports general-purpose computing; rather than a single category of application. Users get a choice of language, but cannot control the underlying operating system or runtime. The libraries provide a degree of automatic network configuration and failover/scalability, but require the developer to declaratively specify some application properties in order to do so. Thus, Azure is intermediate between complete application frameworks like AppEngine on the one hand, and hardware virtual machines like EC2 on the other.

Table 4 summarizes how these three classes virtualize computation, storage, and networking. The scattershot offerings of scalable storage suggest that scalable storage with an API comparable in richness to SQL remains an open research problem (see Section 7). Amazon has begun offering Oracle databases hosted on AWS, but the economics and licensing model of this product makes it a less natural fit for Cloud Computing.

Will one model beat out the others in the Cloud Computing space? We can draw an analogy with programming languages and frameworks. Low-level languages such as C and assembly language allow fine control and close communication with the bare metal, but if the developer is writing a Web application, the mechanics of managing sockets, dispatching requests, and so on are cumbersome and tedious to code, even with good libraries. On the other hand, high-level frameworks such as Ruby on Rails make these mechanics invisible to the programmer, but are only useful if the application readily fits the request/reply structure and the abstractions provided by Rails; any deviation requires diving into the framework at best, and may be awkward to code. No reasonable Ruby developer would argue against the superiority of C for certain tasks, and vice versa. Correspondingly, we believe different tasks will result in demand for different classes of utility computing.

Continuing the language analogy, just as high-level languages can be implemented in lower-level ones, highly-managed cloud platforms can be hosted on top of less-managed ones. For example, AppEngine could be hosted on top of Azure or EC2; Azure could be hosted on top of EC2. Of course, AppEngine and Azure each offer proprietary features (AppEngine's scaling, failover and MegaStore data storage) or large, complex API's (Azure's .NET libraries) that have no free implementation, so any attempt to "clone" AppEngine or Azure would require re-implementing those features or API's—a formidable challenge.

Table 4: Examples of Cloud Computing vendors and how each provides virtualized resources (computation, storage, networking) and ensures scalability and high availability of the resources.

	Amazon Web Services	Microsoft Azure	Google AppEngine
Computation model (VM)	• x86 Instruction Set Architecture (ISA) via Xen VM • Computation elasticity allows scalability, but developer must build the machinery, or third party VAR such as RightScale must provide it	• Microsoft Common Language Runtime (CLR) VM; common intermediate form executed in managed environment • Machines are provisioned based on declarative descriptions (e.g. which "roles" can be replicated); automatic load balancing	• Predefined application structure and framework; programmer-provided "handlers" written in Python, all persistent state stored in MegaStore (outside Python code) • Automatic scaling up and down of computation and storage; network and server failover; all consistent with 3-tier Web app structure
Storage model	• Range of models from block store (EBS) to augmented key/blob store (SimpleDB) • Automatic scaling varies from no scaling or sharing (EBS) to fully automatic (SimpleDB, S3), depending on which model used • Consistency guarantees vary widely depending on which model used • APIs vary from standardized (EBS) to proprietary	• SQL Data Services (restricted view of SQL Server) • Azure storage service	•MegaStore/BigTable
Networking model	• Declarative specification of IP-level topology; internal placement details concealed • Security Groups enable restricting which nodes may communicate • Availability zones provide abstraction of independent network failure • Elastic IP addresses provide persistently routable network name	• Automatic based on programmer's declarative descriptions of app components (roles)	• Fixed topology to accommodate 3-tier Web app structure • Scaling up and down is automatic and programmer-invisible

6 Cloud Computing Economics

In this section we make some observations about Cloud Computing economic models:

- In deciding whether hosting a service in the cloud makes sense over the long term, we argue that the fine-grained economic models enabled by Cloud Computing make tradeoff decisions more fluid, and in particular the elasticity offered by clouds serves to transfer risk.
- As well, although hardware resource costs continue to decline, they do so at variable rates; for example, computing and storage costs are falling faster than WAN costs. Cloud Computing can track these changes—and potentially pass them through to the customer—more effectively than building one's own datacenter, resulting in a closer match of expenditure to actual resource usage.
- In making the decision about whether to move an existing service to the cloud, one must additionally examine the expected average and peak resource utilization, especially if the application may have highly variable spikes in resource demand; the practical limits on real-world utilization of purchased equipment; and various operational costs that vary depending on the type of cloud environment being considered.

6.1 Elasticity: Shifting the Risk

Although the economic appeal of Cloud Computing is often described as "converting capital expenses to operating expenses" (CapEx to OpEx), we believe the phrase "pay as you go" more directly captures the economic benefit to the buyer. Hours purchased via Cloud Computing can be distributed non-uniformly in time (e.g., use 100 server-hours today and no server-hours tomorrow, and still pay only for what you use); in the networking community, this way of selling bandwidth is already known as *usage-based pricing*. [3] In addition, the absence of up-front capital expense allows capital to be redirected to core business investment.

Therefore, even though Amazon's pay-as-you-go pricing (for example) could be more expensive than buying and depreciating a comparable server over the same period, we argue that the cost is outweighed by the extremely important Cloud Computing economic benefits of *elasticity* and *transference of risk*, especially the risks of overprovisioning (underutilization) and underprovisioning (saturation).

We start with elasticity. The key observation is that Cloud Computing's ability to add *or remove* resources at a fine grain (one server at a time with EC2) and with a lead time of minutes rather than weeks allows matching resources to workload much more closely. Real world estimates of server utilization in datacenters range from 5% to 20% [37, 38]. This may sound shockingly low, but it is consistent with the observation that for many services the peak workload exceeds the average by factors of 2 to 10. Few users deliberately provision for *less* than the expected peak, and therefore they must provision for the peak and allow the resources to remain idle at nonpeak times. The more pronounced the variation, the more the waste. A simple example demonstrates how elasticity allows reducing this waste and can therefore more than compensate for the potentially higher cost per server-hour of paying-as-you-go vs. buying.

> **Example: Elasticity.** Assume our service has a predictable daily demand where the peak requires 500 servers at noon but the trough requires only 100 servers at midnight, as shown in Figure 2(a). As long as the *average* utilization over a whole day is 300 servers, the actual utilization over the whole day (shaded area under the curve) is $300 \times 24 = 7200$ server-hours; but since we must provision to the peak of 500 servers, we pay for $500 \times 24 = 12000$ server-hours, a factor of 1.7 more than what is needed. Therefore, as long as the pay-as-you-go cost per server-hour over 3 years[4] is less than 1.7 times the cost of buying the server, we can save money using utility computing.

In fact, the above example *underestimates* the benefits of elasticity, because in addition to simple diurnal patterns, most nontrivial services also experience seasonal or other periodic demand variation (e.g., e-commerce peaks in December and photo sharing sites peak after holidays) as well as some *unexpected* demand bursts due to external events (e.g., news events). Since it can take weeks to acquire and rack new equipment, the only way to handle such spikes is to provision for them in advance. We already saw that even if service operators predict the spike sizes correctly, capacity is wasted, and if they overestimate the spike they provision for, it's even worse.

They may also underestimate the spike (Figure 2(b)), however, accidentally turning away excess users. While the monetary effects of overprovisioning are easily measured, those of underprovisioning are harder to measure yet potentially equally serious: not only do rejected users generate zero revenue, they may never come back due to poor service. Figure 2(c) aims to capture this behavior: users will desert an underprovisioned service until the peak user

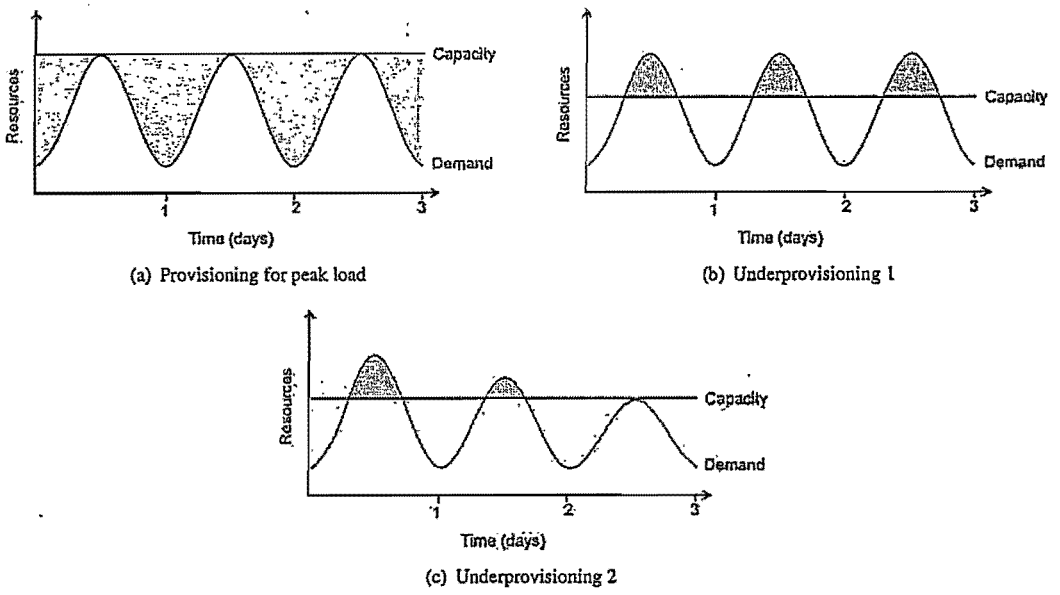


Figure 2: (a) Even if peak load can be correctly anticipated, without elasticity we waste resources (shaded area) during nonpeak times. (b) Underprovisioning case 1: potential revenue from users not served (shaded area) is sacrificed. (c) Underprovisioning case 2: some users desert the site permanently after experiencing poor service; this attrition and possible negative press result in a permanent loss of a portion of the revenue stream.

load equals the datacenter's usable capacity, at which point users again receive acceptable service, but with fewer potential users.

Example: Transferring risks. Suppose but 10% of users who receive poor service due to underprovisioning are "permanently lost" opportunities, i.e. users who would have remained regular visitors with a better experience. The site is initially provisioned to handle an expected peak of 400,000 users (1000 users per server × 400 servers), but unexpected positive press drives 500,000 users in the first hour. Of the 100,000 who are turned away or receive bad service, by our assumption 10,000 of them are permanently lost, leaving an active user base of 390,000. The next hour sees 250,000 new unique users. The first 10,000 do fine, but the site is still over capacity by 240,000 users. This results in 24,000 additional defections, leaving 376,000 permanent users. If this pattern continues, after lg 500000 or 19 hours, the number of new users will approach zero and the site will be at capacity in steady state. Clearly, the service operator has collected less than 400,000 users' worth of steady revenue during those 19 hours, however, again illustrating the underutilization argument —to say nothing of the bad reputation from the disgruntled users.

Do such scenarios really occur in practice? When Animoto [4] made its service available via Facebook, it experienced a demand surge that resulted in growing from 50 servers to 3500 servers in three days. Even if the average utilization of each server was low, no one could have foreseen that resource needs would suddenly double every 12 hours for 3 days. After the peak subsided, traffic fell to a level that was well below the peak. So in this real world example, scale-up elasticity was not a cost optimization but an operational requirement, and scale-down elasticity allowed the steady-state expenditure to more closely match the steady-state workload.

Elasticity is valuable to established companies as well as startups. For example, Target, the nation's second largest retailer, uses AWS for the Target.com website. While other retailers had severe performance problems and intermittent unavailability on "Black Friday" (November 28), Target's and Amazon's sites were just slower by about 50%. [5] Similarly, Salesforce.com hosts customers ranging from 2 seat to 40,000+ seat customers.

Even less-dramatic cases suffice to illustrate this key benefit of Cloud Computing: the risk of mis-estimating workload is shifted from the service operator to the cloud vendor. The cloud vendor may charge a premium (reflected as a higher use cost per server-hour compared to the 3-year purchase cost) for assuming this risk. We propose the following simple equation that generalizes all of the above cases. We assume the Cloud Computing vendor employs

usage-based pricing, in which customers pay proportionally to the amount of time and the amount of resources they use. While some argue for more sophisticated pricing models for infrastructure services [28, 6, 40], we believe usage-based pricing will persist because it is simpler and more transparent, as demonstrated by its wide use by "real" utilities such as electricity and gas companies. Similarly, we assume that the customer's revenue is directly proportional to the total number of user-hours. This assumption is consistent with the ad-supported revenue model in which the number of ads served is roughly proportional to the total visit time spent by end users on the service.

$$\text{UserHours}_{\text{cloud}} \times (\text{revenue} - \text{Cost}_{\text{cloud}}) \geq \text{UserHours}_{\text{datacenter}} \times \left(\text{revenue} - \frac{\text{Cost}_{\text{datacenter}}}{\text{Utilization}}\right) \quad (2)$$

The left-hand side multiplies the net revenue per user-hour (revenue realized per user-hour minus cost of paying Cloud Computing per user-hour) by the number of user-hours, giving the expected profit from using Cloud Computing. The right-hand side performs the same calculation for a fixed-capacity datacenter by factoring in the average utilization, including nonpeak workloads. Whichever side is greater represents the opportunity for higher profit.

Apparently, if Utilization = 1.0 (the datacenter equipment is 100% utilized), the two sides of the equation look the same. However, basic queueing theory tells us that as utilization approaches 1.0, system response time approaches infinity. In practice, the *usable* capacity of a datacenter (without compromising service) is typically 0.6 to 0.8.[6] Whereas a datacenter must necessarily overprovision to account for this "overhead," the cloud vendor can simply factor it into $\text{Cost}_{\text{cloud}}$. (This overhead explains why we use the phrase "pay-as-you-go" rather than rent or lease for utility computing. The latter phrases include this unusable overhead, while the former doesn't. Hence, even if you lease a 100 Mbits/second Internet link, you can likely use only 60 to 80 Mbits/second in practice.)

The equation makes clear that the common element in all of our examples is the ability to control the cost per user-hour of operating the service. In Example 1, the cost per user-hour without elasticity was high because of resources sitting idle—higher costs but same number of user-hours. The same thing happens when over-estimation of demand results in provisioning for workload that doesn't materialize. In Example 2, the cost per user-hour increased as a result of underestimating a spike and having to turn users away: Since some fraction of those users never return, the fixed costs stay the same but are now amortized over fewer user-hours. This illustrates fundamental limitations of the "buy" model in the face of any nontrivial burstiness in the workload.

Finally, there are two additional benefits to the Cloud Computing user that result from being able to change their resource usage on the scale of hours rather than years. First, unexpectedly scaling down (disposing of temporarily-underutilized equipment)—for example, due to a business slowdown, or ironically due to improved software efficiency—normally carries a financial penalty. With 3-year depreciation, a $2,100 server decommissioned after 1 year of operation represents a "penalty" of $1,400. Cloud Computing eliminates this penalty.

Second, technology trends suggest that over the useful lifetime of some purchased equipment, hardware costs will fall and new hardware and software technologies will become available. Cloud providers, who already enjoy economy-of-scale buying power as described in Section 3, can potentially pass on some of these savings to their customers. Indeed, heavy users of AWS saw storage costs fall 20% and networking costs fall 50% over the last 2.5 years, and the addition of nine new services or features to AWS over less than one year. [7] If new technologies or pricing plans become available to a cloud vendor, existing applications and customers can potentially benefit from them immediately, without incurring a capital expense. In less than two years, Amazon Web Services increased the number of different types of compute servers ("instances") from one to five, and in less than one year they added seven new infrastructure services and two new operational support options. [8]

6.2 Comparing Costs: Should I Move to the Cloud?

Whereas the previous section tried to quantify the economic value of specific Cloud Computing benefits such as elasticity, this section tackles an equally important but larger question: Is it more economical to move my existing datacenter-hosted service to the cloud, or to keep it in a datacenter?

Table 5 updates Gray's 2003 cost data [21] to 2008, allowing us to track the rate of change of key technologies for Cloud Computing for the last 5 years. Note that, as expected, wide-area networking costs have improved the least in 5 years, by less than a factor of 3. While computing costs have improved the most in 5 years, the ability to use the extra computing power is based on the assumption that programs can utilize all the cores on both sockets in the computer. This assumption is likely more true for Utility Computing, with many Virtual Machines serving thousands to millions of customers, than it is for programs inside the datacenter of a single company.

To facilitate calculations, Gray calculated what $1 bought in 2003. Table 5 shows his numbers vs. 2008 and compares to EC2/S3 charges. At first glance, it appears that a given dollar will go further if used to purchase hardware in 2008 than to pay for use of that same hardware. However, this simple analysis glosses over several important factors.

Pay separately per resource. Most applications do not make equal use of computation, storage, and network bandwidth; some are CPU-bound, others network-bound, and so on, and may saturate one resource while underutiliz-

Table 5: We update Gray's costs of computing resources from 2003 to 2008, normalize to what $1 could buy in 2003 vs. 2008, and compare to the cost of paying per use of $1 worth of resources on AWS at 2008 prices.

	WAN bandwidth/mo.	CPU hours (all cores)	disk storage
Item in 2003	1 Mbps WAN link	2 GHz CPU, 2 GB DRAM	200 GB disk, 50 Mb/s transfer rate
Cost in 2003	$100/mo.	$2000	$200
$1 buys in 2003	1 GB	8 CPU hours	1 GB
Item in 2008	100 Mbps WAN link	2 GHz, 2 sockets, 4 cores/socket, 4 GB DRAM	1 TB disk, 115 MB/s sustained transfer
Cost in 2008	$3600/mo.	$1000	$100
$1 buys in 2008	2.7 GB	128 CPU hours	10 GB
cost/performance improvement	2.7x	16x	10x
Cost to rent $1 worth on AWS in 2008	$0.27–$0.40 ($0.10–$0.15/GB × 3 GB)	$2.56 (128× 2 VM's@$0.10 each)	$1.20–$1.50 ($0.12–$0.15/GB-month × 10 GB)

ing others. Pay-as-you-go Cloud Computing can charge the application separately for each type of resource, reducing the waste of underutilization. While the exact savings depends on the application, suppose the CPU is only 50% utilized while the network is at capacity; then in a datacenter you are effectively paying for double the number of CPU cycles actually being used. So rather than saying it costs $2.56 to rent only $1 worth of CPU, it would be more accurate to say it costs $2.56 to rent $2 worth of CPU. As a side note, AWS's prices for wide-area networking are actually more competitive than what a medium-sized company would pay for the same bandwidth.

Power, cooling and physical plant costs. The costs of power, cooling, and the amortized cost of the building are missing from our simple analyses so far. Hamilton estimates that the costs of CPU, storage and bandwidth roughly double when those costs are amortized over the building's lifetime [23, 26]. Using this estimate, buying 128 hours of CPU in 2008 really costs $2 rather than $1, compared to $2.56 on EC2. Similarly, 10 GB of disk space costs $2 rather than $1, compared to $1.20–$1.50 per month on S3. Lastly, S3 actually replicates the data at least 3 times for durability and performance, ensure durability, and will replicate it further for performance is there is high demand for the data. That means the costs are $6.00 when purchasing vs. $1.20 to $1.50 per month on S3.

Operations costs. Today, hardware operations costs are very low—rebooting servers is easy (e.g., IP addressable power strips, separate out of band controllers, and so on) and minimally trained staff can replace broken components at the rack or server level. On one hand, since Utility Computing uses virtual machines instead of physical machines, from the cloud user's point of view these tasks are shifted to the cloud provider. On the other hand, depending on the level of virtualization, much of the software management costs may remain—upgrades, applying patches, and so on. Returning to the "managed vs. unmanaged" discussion of Section 5, we believe these costs will be lower for managed environments (e.g. Microsoft Azure, Google AppEngine, Force.com) than for hardware-level utility computing (e.g. Amazon EC2), but it seems hard to quantify these benefits in a way that many would agree with.

With the above caveats in mind, here is a simple example of deciding whether to move a service into the cloud.

Example: Moving to cloud. Suppose a biology lab creates 500 GB of new data for every wet lab experiment. A computer the speed of one EC2 instance takes 2 hours per GB to process the new data. The lab has the equivalent 20 instances locally, so the time to evaluate the experiment is 500 × 2/20 or 50 hours. They could process it in a single hour on 1000 instances at AWS. The cost to process one experiment would be just 1000 × $0.10 or $100 in computation and another 500 × $0.10 or $50 in network transfer fees. So far, so good. They measure the transfer rate from the lab to AWS at 20 Mbits/second. [19] The transfer time is $(500GB \times 1000MB/GB \times 8bits/Byte)/20Mbits/sec = 4,000,000/20 = 200,000$ seconds or more than 55 hours. Thus, it takes 50 hours locally vs. 55 + 1 or 56 hours on AWS, so they don't move to the cloud. (The next section offers an opportunity on how to overcome the transfer delay obstacle.)

A related issue is the software complexity and costs of (partial or full) migrating data from a legacy enterprise application into the Cloud. While migration is a one-time task, the amount of effort can be significant and it needs to be considered as a factor in deciding to use Cloud Computing. This task is already spawning new business opportunities for companies that provide data integration across public and private Clouds.

Table 6: Top 10 Obstacles to and Opportunities for Adoption and Growth of Cloud Computing.

	Obstacle	Opportunity
1	Availability of Service	Use Multiple Cloud Providers to provide Business Continuity; Use Elasticity to Defend Against DDOS attacks
2	Data Lock-In	Standardize APIs; Make compatible software available to enable Surge Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, and Firewalls; Accommodate National Laws via Geographical Data Storage
4	Data Transfer Bottlenecks	FedExing Disks; Data Backup/Archival; Lower WAN Router Costs; Higher Bandwidth LAN Switches
5	Performance Unpredictability	Improved Virtual Machine Support; Flash Memory; Gang Scheduling VMs for HPC apps
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large-Scale Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on Machine Learning; Snapshots to encourage Cloud Computing Conservationism
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses; Bulk use sales

7 Top 10 Obstacles and Opportunities for Cloud Computing

In this section, we offer a ranked list of obstacles to the growth of Cloud Computing. Each obstacle is paired with an *opportunity*—our thoughts on how to overcome the obstacle, ranging from straightforward product development to major research projects. Table 6 summarizes our top ten obstacles and opportunities. The first three are technical obstacles to the *adoption* of Cloud Computing, the next five are technical obstacles to the *growth* of Cloud Computing once it has been adopted, and the last two are *policy and business* obstacles to the *adoption* of Cloud Computing.

Number 1 Obstacle: Availability of a Service

Organizations worry about whether Utility Computing services will have adequate availability, and this makes some wary of Cloud Computing. Ironically, existing SaaS products have set a high standard in this regard. Google Search is effectively the dial tone of the Internet: if people went to Google for search and it wasn't available, they would think the Internet was down. Users expect similar availability from new services, which is hard to do. Table 7 shows recorded outages for Amazon Simple Storage Service (S3), AppEngine and Gmail in 2008, and explanations for the outages. Note that despite the negative publicity due to these outages, few enterprise IT infrastructures are as good.

Table 7: Outages in AWS, AppEngine, and Gmail

Service and Outage	Duration	Date
S3 outage: authentication service overload leading to unavailability [39]	2 hours	2/15/08
S3 outage: Single bit error leading to gossip protocol blowup. [41]	6-8 hours	7/20/08
AppEngine partial outage: programming error [43]	5 hours	6/17/08
Gmail: site unavailable due to outage in contacts system [29]	1.5 hours	8/11/08

Just as large Internet service providers use multiple network providers so that failure by a single company will not take them off the air, we believe the only plausible solution to very high availability is multiple Cloud Computing providers. The high-availability computing community has long followed the mantra "no single source of failure," yet the management of a Cloud Computing service by a single company is in fact a single point of failure. Even if the company has multiple datacenters in different geographic regions using different network providers, it may have common software infrastructure and accounting systems, or the company may even go out of business. Large customers will be reluctant to migrate to Cloud Computing without a business-continuity strategy for such situations. We believe the best chance for independent software stacks is for them to be provided by different companies, as it has been difficult for one company to justify creating and maintain two stacks in the name of software dependability.

Another availability obstacle is Distributed Denial of Service (DDoS) attacks. Criminals threaten to cut off the incomes of SaaS providers by making their service unavailable, extorting $10,000 to $50,000 payments to prevent the launch of a DDoS attack. Such attacks typically use large "botnets" that rent bots on the black market for $0.03 per

bot (simulated bogus user) per week [36]. Utility Computing offers SaaS providers the opportunity to defend against DDoS attacks by using quick scale-up. Suppose an EC2 instance can handle 500 bots, and an attack is launched that generates an extra 1 GB/second of bogus network bandwidth and 500,000 bots. At $0.03 per bot, such an attack would cost the attacker $15,000 invested up front. At AWS's current prices, the attack would cost the victim an extra $360 per hour in network bandwidth and an extra $100 per hour (1,000 instances) of computation. The attack would therefore have to last 32 hours in order to cost the potential victim more than it would the blackmailer. A botnet attack this long may be difficult to sustain, since the longer an attack lasts the easier it is to uncover and defend against, and the attacking bots could not be immediately re-used for other attacks on the same provider. As with elasticity, Cloud Computing shifts the attack target from the SaaS provider to the Utility Computing provider, who can more readily absorb it and (as we argued in Section 3) is also likely to have already DDoS protection as a core competency.

Number 2 Obstacle: Data Lock-In

Software stacks have improved interoperability among platforms, but the APIs for Cloud Computing itself are still essentially proprietary, or at least have not been the subject of active standardization. Thus, customers cannot easily extract their data and programs from one site to run on another. Concern about the difficult of extracting data from the cloud is preventing some organizations from adopting Cloud Computing. Customer lock-in may be attractive to Cloud Computing providers, but Cloud Computing users are vulnerable to price increases (as Stallman warned), to reliability problems, or even to providers going out of business.

For example, an online storage service called The Linkup shut down on August 8, 2008 after losing access as much as 45% of customer data [12]. The Linkup, in turn, had relied on the online storage service Nirvanix to store customer data, and now there is finger pointing between the two organizations as to why customer data was lost. Meanwhile, The Linkup's 20,000 users were told the service was no longer available and were urged to try out another storage site.

The obvious solution is to standardize the APIs so that a SaaS developer could deploy services and data across multiple Cloud Computing providers so that the failure of a single company would not take all copies of customer data with it. The obvious fear is that this would lead to a "race-to-the-bottom" of cloud pricing and flatten the profits of Cloud Computing providers. We offer two arguments to allay this fear.

First, the quality of a service matters as well as the price, so customers will not necessarily jump to the lowest cost service. Some Internet Service Providers today cost a factor of ten more than others because they are more dependable and offer extra services to improve usability.

Second, in addition to mitigating data lock-in concerns, standardization of APIs enables a new usage model in which the same software infrastructure can be used in a Private Cloud and in a Public Cloud. [9] Such an option could enable "Surge Computing," in which the public Cloud is used to capture the extra tasks that cannot be easily run in the datacenter (or private cloud) due to temporarily heavy workloads. [10]

Number 3 Obstacle: Data Confidentiality and Auditability

"My sensitive corporate data will never be in the cloud." Anecdotally we have heard this repeated multiple times. Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. There are also requirements for auditability, in the sense of Sarbanes-Oxley and Health and Human Services Health Insurance Portability and Accountability Act (HIPAA) regulations that must be provided for corporate data to be moved to the cloud.

We believe that there are no fundamental obstacles to making a cloud-computing environment as secure as the vast majority of in-house IT environments, and that many of the obstacles can be overcome immediately with well-understood technologies such as encrypted storage, Virtual Local Area Networks, and network middleboxes (e.g. firewalls, packet filters). For example, encrypting data before placing it in a Cloud may be even more secure than unencrypted data in a local data center; this approach was successfully used by TC3, a healthcare company with access to sensitive patient records and healthcare claims, when moving their HIPAA-compliant application to AWS [2].

Similarly, auditability could be added as an additional layer beyond the reach of the virtualized guest OS (or virtualized application environment), providing facilities arguably more secure than those built into the applications themselves and centralizing the software responsibilities related to confidentiality and auditability into a single logical layer. Such a new feature reinforces the Cloud Computing perspective of changing our focus from specific hardware to the virtualized capabilities being provided.

A related concern is that many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries. Similarly, some businesses may not like the ability of a country to get access to their data via the court system; for example, a European customer might be concerned about using SaaS in the United States given the USA PATRIOT Act.

Cloud Computing gives SaaS providers and SaaS users greater freedom to place their storage. For example, Amazon provides S3 services located physically in the United States and in Europe, allowing providers to keep data in whichever they choose. With AWS regions, a simple configuration change avoids the need to find and negotiate with a hosting provider overseas.

Number 4 Obstacle: Data Transfer Bottlenecks

Applications continue to become more data-intensive. If we assume applications may be "pulled apart" across the boundaries of clouds, this may complicate data placement and transport. At $100 to $150 per terabyte transferred, these costs can quickly add up, making data transfer costs an important issue. Cloud users and cloud providers have to think about the implications of placement and traffic at every level of the system if they want to minimize costs. This kind of reasoning can be seen in Amazon's development of their new Cloudfront service.

One opportunity to overcome the high cost of Internet transfers is to ship disks. Jim Gray found that the cheapest way to send a lot of data is to physically send disks or even whole computers via overnight delivery services [22]. Although there are no guarantees from the manufacturers of disks or computers that you can reliably ship data that way, he experienced only one failure in about 400 attempts (and even this could be mitigated by shipping extra disks with redundant data in a RAID-like manner).

To quantify the argument, assume that we want to ship 10 TB from U.C. Berkeley to Amazon in Seattle, Washington. Garfinkel measured bandwidth to S3 from three sites and found an average write bandwidth of 5 to 18 Mbits/second. [19] Suppose we get 20 Mbit/sec over a WAN link. It would take
$10 * 10^{12}$ Bytes / $(20 \times 10^6$ bits/second$) = (8 \times 10^{13})/(2 \times 10^7)$ seconds = 4,000,000 seconds,
which is more than 45 days. Amazon would also charge you $1000 in network transfer fees when it received the data.

If we instead sent ten 1 TB disks via overnight shipping, it would take less than a day to transfer 10 TB and the cost would be roughly $400, an effective bandwidth of about 1500 Mbit/sec.[11] Thus, "Netflix for Cloud Computing" could halve costs of bulk transfers into the cloud but more importantly reduce latency by a factor of 45.

Returning to the biology lab example from Section 6, it would take about 1 hour to write a disk, 16 hours to FedEx a disk, about 1 hour to read 500 GB, and then 1 hour to process it. Thus, the time to process the experiment would be 20 hours instead of 50, and the cost is would be around $200 per experiment, so they decide to move to the cloud after all. As disk capacity and cost-per-gigabyte are growing much faster than network cost-performance—10X vs. less than 3X in the last 5 years according to Table 5—the FedEx disk option for large data transfers will get more attractive each year.

A second opportunity is to find other reasons to make it attractive to keep data in the cloud, for once data is in the cloud for any reason it may no longer be a bottleneck and may enable new services that could drive the purchase of Cloud Computing cycles. Amazon recently began hosting large public datasets (e.g. US Census data) for free on S3; since there is no charge to transfer data between S3 and EC2, these datasets might "attract" EC2 cycles. As another example, consider off-site archival and backup services. Since companies like Amazon, Google, and Microsoft likely send much more data than they receive, the cost of ingress bandwidth could be much less. Therefore, for example, if weekly full backups are moved by shipping physical disks and compressed daily incremental backups are sent over the network, Cloud Computing might be able to offer an affordable off-premise backup service. Once archived data is in the cloud, new services become possible that could result in selling more Cloud Computing cycles, such as creating searchable indices of all your archival data or performing image recognition on all your archived photos to group them according to who appears in each photo.[12]

A third, more radical opportunity is to try to reduce the cost of WAN bandwidth more quickly. One estimate is that two-thirds of the cost of WAN bandwidth is the cost of the high-end routers, whereas only one-third is the fiber cost [27]. Researchers are exploring simpler routers built from commodity components with centralized control as a low-cost alternative to the high-end distributed routers [33]. If such technology were deployed by WAN providers, we could see WAN costs dropping more quickly than they have historically.

In addition to WAN bandwidth being a bottleneck, intra-cloud networking technology may be a performance bottleneck as well. Today inside the datacenter, typically 20-80 processing nodes within a rack are connected via a top-of-rack switch to a second level aggregation switch. These in turn are connected via routers to storage area networks and wide-area connectivity, such as the Internet or inter-datacenter WANs. Inexpensive 1 Gigabit Ethernet (1GbE) is universally deployed at the lower levels of aggregation. This bandwidth can represent a performance bottleneck for inter-node processing patterns that burst packets across the interconnect, such as the shuffle step that occurs between Map and Reduce producing. Another set of batch applications that need higher bandwidth is high performance computing applications; lack of bandwidth is one reason few scientists using Cloud Computing.

10 Gigabit Ethernet is typically used for the aggregation links in cloud networks, but is currently too expensive to deploy for individual servers (about $1000 for a 10 GbE server connection today, vs. $100 for a 1GbE connection). However, as the cost per 10 GbE server connections is expected to drop to less than $200 in 2010, it will gain

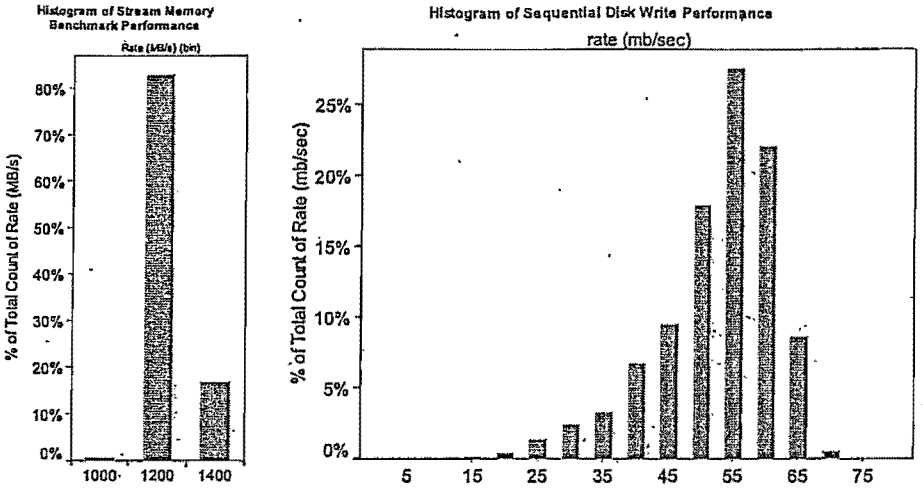


Figure 3: (a) Memory benchmark performance on 75 Virtual Machines running the STREAM benchmark on left and (b) Disk performance writing 1 GB files on 75 Virtual Machines on right.

widespread deployment inside the cloud since it has the highly desirable effect of reducing data transfer latencies and network contention. This in turn enables more cores and virtual machines per physical server node by scaling up the network. Also in 2010, 40 GbE and 100 GbE will appear for the higher aggregation layers [10].

Number 5 Obstacle: Performance Unpredictability

Our experience is that multiple Virtual Machines can share CPUs and main memory surprisingly well in Cloud Computing, but that I/O sharing is more problematic. Figure 3(a) shows the average memory bandwidth for 75 EC2 instances running the STREAM memory benchmark [32]. The mean bandwidth is 1355 MBytes per second, with a standard deviation of just 52 MBytes/sec, less than 4% of the mean. Figure 3(b) shows the average disk bandwidth for 75 EC2 instances each writing 1 GB files to local disk. The mean disk write bandwidth is nearly 55 MBytes per second with a standard deviation of a little over 9 MBytes/sec, more than 16% of the mean. This demonstrates the problem of I/O interference between virtual machines.

One opportunity is to improve architectures and operating systems to efficiently virtualize interrupts and I/O channels. Technologies such as PCIexpress are difficult to virtualize, but they are critical to the cloud. One reason to be hopeful is that IBM mainframes and operating systems largely overcame these problems in the 1980s, so we have successful examples from which to learn.

Another possibility is that flash memory will decrease I/O interference. Flash is semiconductor memory that preserves information when powered off like mechanical hard disks, but since it has no moving parts, it is much faster to access (microseconds vs. milliseconds) and uses less energy. Flash memory can sustain many more I/Os per second per gigabyte of storage than disks, so multiple virtual machines with conflicting random I/O workloads could coexist better on the same physical computer without the interference we see with mechanical disks. The lack of interference that we see with semiconductor main memory in Figure 3(a) might extend to semiconductor storage as well, thereby increasing the number of applications that can run well on VMs and thus share a single computer. This advance could lower costs to Cloud Computing providers, and eventually to Cloud Computing consumers.

Another unpredictability obstacle concerns the scheduling of virtual machines for some classes of batch processing programs, specifically for high performance computing. Given that high-performance computing is used to justify Government purchases of $100M supercomputer centers with 10,000 to 1,000,000 processors, there certainly are many tasks with parallelism that can benefit from elastic computing. Cost associativity means that there is no cost penalty for using 20 times as much computing for $1/20^{th}$ the time. Potential applications that could benefit include those with very high potential financial returns—financial analysis, petroleum exploration, movie animation—and could easily justify paying a modest premium for a 20x speedup. One estimate is that a third of today's server market is high-performance computing [10].

The obstacle to attracting HPC is not the use of clusters; most parallel computing today is done in large clusters using the message-passing interface MPI. The problem is that many HPC applications need to ensure that all the threads of a program are running simultaneously, and today's virtual machines and operating systems do not provide

a programmer-visible way to ensure this. Thus, the opportunity to overcome this obstacle is to offer something like "gang scheduling" for Cloud Computing.[13]

Number 6 Obstacle: Scalable Storage

Early in this paper, we identified three properties whose combination gives Cloud Computing its appeal: short-term usage (which implies scaling down as well as up when resources are no longer needed), no up-front cost, and infinite capacity on-demand. While it's straightforward what this means when applied to computation, it's less obvious how to apply it to persistent storage.

As Table 4 shows, there have been many attempts to answer this question, varying in the richness of the query and storage API's, the performance guarantees offered, and the complexity of data structures that are directly supported by the storage system (e.g., schema-less blobs vs. column-oriented storage).[14] The opportunity, which is still an open research problem, is to create a storage system would not only meet these needs but combine them with the cloud advantages of scaling arbitrarily up and down on-demand, as well as meeting programmer expectations in regard to resource management for scalability, data durability, and high availability.

Number 7 Obstacle: Bugs in Large-Scale Distributed Systems

One of the difficult challenges in Cloud Computing is removing errors in these very large scale distributed systems. A common occurrence is that these bugs cannot be reproduced in smaller configurations, so the debugging must occur at scale in the production datacenters.

One opportunity may be the reliance on virtual machines in Cloud Computing. Many traditional SaaS providers developed their infrastructure without using VMs, either because they preceded the recent popularity of VMs or because they felt they could not afford the performance hit of VMs. Since VMs are *de rigueur* in Utility Computing, that level of virtualization may make it possible to capture valuable information in ways that are implausible without VMs.

Number 8 Obstacle: Scaling Quickly

Pay-as-you-go certainly applies to storage and to network bandwidth, both of which count bytes used. Computation is slightly different, depending on the virtualization level. Google AppEngine automatically scales in response to load increases and decreases, and users are charged by the cycles used. AWS charges by the hour for the number of instances you occupy, even if your machine is idle.

The opportunity is then to automatically scale quickly up and down in response to load in order to save money, but without violating service level agreements. Indeed, one RAD Lab focus is the pervasive and aggressive use of statistical machine learning as a diagnostic and predictive tool that would allow dynamic scaling, automatic reaction to performance and correctness problems, and generally automatic management of many aspects of these systems.

Another reason for scaling is to conserve resources as well as money. Since an idle computer uses about two-thirds of the power of a busy computer, careful use of resources could reduce the impact of datacenters on the environment, which is currently receiving a great deal of negative attention. Cloud Computing providers already perform careful and low overhead accounting of resource consumption. By imposing per-hour and per-byte costs, utility computing encourages programmers to pay attention to efficiency (i.e., releasing and acquiring resources only when necessary), and allows more direct measurement of operational and development inefficiencies.

Being aware of costs is the first step to conservation, but the hassles of configuration make it tempting to leave machines idle overnight so that nothing has to be done to get started when developers return to work the next day. A fast and easy-to-use snapshot/restart tool might further encourage conservation of computing resources.

Number 9 Obstacle: Reputation Fate Sharing

Reputations do not virtualize well. One customer's bad behavior can affect the reputation of the cloud as a whole. For instance, blacklisting of EC2 IP addresses [31] by spam-prevention services may limit which applications can be effectively hosted. An opportunity would be to create reputation-guarding services similar to the "trusted email" services currently offered (for a fee) to services hosted on smaller ISP's, which experience a microcosm of this problem.

Another legal issue is the question of transfer of legal liability—Cloud Computing providers would want legal liability to remain with the customer and not be transferred to them (i.e., the company sending the spam should be held liable, not Amazon).

Number 10 Obstacle: Software Licensing

Current software licenses commonly restrict the computers on which the software can run. Users pay for the software and then pay an annual maintenance fee. Indeed, SAP announced that it would increase its annual maintenance fee to at least 22% of the purchase price of the software, which is comparable to Oracle's pricing [38]. Hence, many cloud computing providers originally relied on open source software in part because the licensing model for commercial software is not a good match to Utility Computing.

The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit Cloud Computing. For example, Microsoft and Amazon now offer pay-as-you-go software licensing for Windows Server and Windows SQL Server on EC2. An EC2 instance running Microsoft Windows costs $0.15 per hour instead of the traditional $0.10 per hour of the open source version.[15]

A related obstacle is encouraging sales forces of software companies to sell products into Cloud Computing. Pay-as-you-go seems incompatible with the quarterly sales tracking used to measure effectiveness, which is based on one-time purchases. The opportunity for cloud providers is simply to offer prepaid plans for bulk use that can be sold at discount. For example, Oracle sales people might sell 100,000 instance hours using Oracle that can be used over the next two years at a cost less than is the customer were to purchase 100,000 hours on their own. They could then meet their quarterly quotas and make their commissions from cloud sales as well as from traditional software sales, potentially converting this customer-facing part of a company from naysayers into advocates of cloud computing.

8 Conclusion and Questions about the Clouds of Tomorrow

The long dreamed vision of computing as a utility is finally emerging. The elasticity of a utility matches the need of businesses providing services directly to customers over the Internet, as workloads can grow (and shrink) far faster than 20 years ago. It used to take years to grow a business to several million customers – now it can happen in months.

From the cloud provider's view, the construction of very large datacenters at low cost sites using commodity computing, storage, and networking uncovered the possibility of selling those resources on a pay-as-you-go model below the costs of many medium-sized datacenters, while making a profit by statistically multiplexing among a large group of customers. From the cloud user's view, it would be as startling for a new software startup to build its own datacenter as it would for a hardware startup to build its own fabrication line. In addition to startups, many other established organizations take advantage of the elasticity of Cloud Computing regularly, including newspapers like the Washington Post, movie companies like Pixar, and universities like ours. Our lab has benefited substantially from the ability to complete research by conference deadlines and adjust resources over the semester to accommodate course deadlines. As Cloud Computing users, we were relieved of dealing with the twin dangers of over-provisioning and under-provisioning our internal datacenters.

Some question whether companies accustomed to high-margin businesses, such as ad revenue from search engines and traditional packaged software, can compete in Cloud Computing. First, the question presumes that Cloud Computing is a small margin business based on its low cost. Given the typical utilization of medium-sized datacenters, the potential factors of 5 to 7 in economies of scale, and the further savings in selection of cloud datacenter locations, the apparently low costs offered to cloud users may still be highly profitable to cloud providers. Second, these companies may already have the datacenter, networking, and software infrastructure in place for their mainline businesses, so Cloud Computing represents the opportunity for more income at little extra cost.

Although Cloud Computing providers may run afoul of the obstacles summarized in Table 6, we believe that over the long run providers will successfully navigate these challenges and set an example for others to follow, perhaps by successfully exploiting the opportunities that correspond to those obstacles.

Hence, developers would be wise to design their next generation of systems to be deployed into Cloud Computing. In general, the emphasis should be horizontal scalability to hundreds or thousands of virtual machines over the efficiency of the system on a single virtual machine. There are specific implications as well:

- Applications Software of the future will likely have a piece that runs on clients and a piece that runs in the Cloud. The cloud piece needs to both scale *down* rapidly as well as scale up, which is a new requirement for software systems. The client piece needs to be useful when disconnected from the Cloud, which is not the case for many Web 2.0 applications today. Such software also needs a pay-for-use licensing model to match needs of Cloud Computing.

- Infrastructure Software of the future needs to be cognizant that it is no longer running on bare metal but on virtual machines. Moreover, it needs to have billing built in from the beginning, as it is very difficult to retrofit an accounting system.

- Hardware Systems of the future need to be designed at the scale of a container (at least a dozen racks) rather than at the scale of a single 1U box or single rack, as that is the minimum level at which it will be purchased. Cost of operation will match performance and cost of purchase in importance in the acquisition decision. Hence, they need to strive for *energy proportionality* [9] by making it possible to put into low power mode the idle portions of the memory, storage, and networking, which already happens inside a microprocessor today. Hardware should also be designed assuming that the lowest level software will be virtual machines rather than a single native operating system, and it will need to facilitate flash as a new level of the memory hierarchy between DRAM and disk. Finally, we need improvements in bandwidth and costs for both datacenter switches and WAN routers.

While we are optimistic about the future of Cloud Computing, we would love to look into a crystal ball to see how popular it is and what it will look like in five years:

Change In Technology and Prices Over Time: What will billing units be like for the higher-level virtualization clouds? What will Table 5, tracking the relative prices of different resources, look like? Clearly, the number of cores per chip will increase over time, doubling every two to four years. Flash memory has the potential of adding another relatively fast layer to the classic memory hierarchy; what will be its billing unit? Will technology or business innovations accelerate network bandwidth pricing, which is currently the most slowly-improving technology?

Virtualization Level: Will Cloud Computing be dominated by low-level hardware virtual machines like Amazon EC2, intermediate language offerings like Microsoft Azure, or high-level frameworks like Google AppEngine? Or will we have many virtualization levels that match different applications? Will value-added services by independent companies like RightScale, Heroku, or EngineYard survive in Utility Computing, or will the successful services be entirely co-opted by the Cloud providers? If they do consolidate to a single virtualization layer, will multiple companies embrace a common standard? Will this lead to a race to the bottom in pricing so that it's unattractive to become a Cloud Computing provider, or will they differentiate in services or quality to maintain margins?

Acknowledgments

We all work in the RAD Lab. Its existence is due to the generous support of the founding members Google, Microsoft, and Sun Microsystems and to the affiliate members Amazon Web Services, Cisco Systems, Facebook, Hewlett-Packard, IBM, NEC, Network Appliance, Oracle, Siemens, and VMware; by matching funds from the State of California's MICRO program (grants 06-152, 07-010, 06-148, 07-012, 06-146, 07-009, 06-147, 07-013, 06-149, 06-150, and 07-008) and the University of California Industry/University Cooperative Research Program (UC Discovery) grant COM07-10240; and by the National Science Foundation (grant #CNS-0509559).

We would also like to thank the following people for feedback that improved the alpha draft of this report: Luiz Barroso, Andy Bechtolsheim, John Cheung, David Cheriton, Mike Franklin, James Hamilton, Jeff Hammerbacher, Marvin Theimer, Hal Varian, and Peter Vosshall. For the beta draft, we'd like to thank the following for their comments: Andy Bechtolsheim, Jim Blakely, Paul Hofmann, Kim Keeton, Jim Larus, John Ousterhout, Steve Whittaker, and Feng Zhao.

Notes

[1]The related term "grid computing," from the High Performance Computing community, suggests protocols to offer shared computation and storage over long distances, but those protocols did not lead to a software environment that grew beyond its community. Another phrase found in Cloud Computing papers is *multitenant*, which simply means multiple customers from different companies are using SaaS, so customers and their data need to be protected from each other.

[2]The challenge of disconnected operation is not new to cloud computing; extensive research has examined the problems of disconnected operation, with roots in the Coda filesystem [30] and the Bayou database [18]. We simply point out that satisfactory application-level and protocol-level solutions have been developed and adopted in many domains, including IMAP email, CalDAV calendars, version-control systems such as CVS and Subversion, and recently, Google Gears for JavaScript in-browser applications that can run disconnected. We are confident that similar approaches will develop as demanded by mobile applications that wish to use the cloud.

[3]Usage-based pricing is different from renting. Renting a resource involves paying a negotiated cost to have the resource over some time period, whether or not you use the resource. Pay-as-you-go involves metering usage and charging based on actual use, independently of the time period over which the usage occurs. Amazon AWS rounds up their billing to the nearest server-hour or gigabyte-month, but the associated dollar amounts are small enough (pennies) to make AWS a true pay-as-you-go service.

[4]The most common financial models used in the US allow a capital expense to be depreciated (deducted from tax obligations) linearly over a 3-year period, so we use this figure as an estimate of equipment lifetime in our cost comparisons.

[5]According to statistics collected by Keynote Systems Inc. on Black Friday 2008 (November 28th), Target and Amazon's e-commerce sites were slower on Friday — "a transaction that took 25 seconds last week required about 40 seconds Friday morning" [5].

[6]2nd edition of Hennessy/Patterson had these rules of thumb for storage systems:

- I/O bus < 75%
- Disk bus SCSI < 40% (when attach multiple disks per bus)

- Disk arm seeking < 60%
- Disk IO per second or MB/s < 80% peak

Hence, 60% to 80% is a safe upper bound.

[7]Table 8 shows changes in prices for AWS storage and networking over 2.5 years.

Table 8: Changes in price of AWS S3 storage and networking over time.

Storage	Cost of Data Stored per GB-Month				
Date	< 50 TB	50-100 TB	100-500 TB	> 500 TB	
3/13/06	$0.15	$0.15	$0.15	$0.15	
10/9/08	$0.15	$0.14	$0.13	$0.12	
% Original Price	100%	93%	87%	80%	
Networking	Cost per GB of Wide-Area Networking Traffic				
Date	In	Out: < 10 TB	Out: 10-50 TB	Out: 50-150 TB	Out: >150 TB
3/13/06	$0.20	$0.20	$0.20	$0.20	$0.20
10/31/07	$0.10	$0.18	$0.16	$0.13	$0.13
5/1/08	$0.10	$0.17	$0.13	$0.11	$0.10
% Original Price	50%	85%	65%	55%	50%

[8] Table 9 shows the new services and support options AWS added during 2008, and the date of each introduction. Table 10 shows the different types of AWS compute instances and the date each type was introduced.

Table 9: New AWS Services.

Date	New Service
3-Dec-08	Public Data Sets on AWS Now Available
18-Nov-08	Announcing Amazon CloudFront (Content Distribution Network)
23-Oct-08	Amazon EC2 Running Windows Server Now Available
23-Oct-08	Amazon EC2 Exits Beta and Now Offers a Service Level Agreement
22-Sep-08	Oracle Products Licensed for Amazon Web Services
20-Aug-08	Amazon Elastic Block Store Now Available
5-May-08	OpenSolaris and MySQL Enterprise on Amazon EC2
16-Apr-08	Announcing AWS Premium Support
26-Mar-08	Announcing Elastic IP Addresses and Availability Zones for Amazon EC2

Table 10: Diversity of EC2 instances over time.

Date	Type	Cost/ Hour	Compute Units	DRAM (GB)	Disk (GB)	Compute/$	GB DRAM/$	GB Disk/$
8/24/06	Small	$0.10	1	1.7	160	10	17.0	1600
10/22/07	Large	$0.40	4	7.5	850	10	18.8	2130
10/22/07	Extra Large	$0.80	8	15.0	1690	10	18.8	2110
5/29/08	High-CPU Medium	$0.20	5	1.7	350	25	8.5	1750
5/29/08	High-CPU Extra Large	$0.80	20	7.0	1690	25	8.8	2110

[9]While such standardization can occur for the full spectrum of utility computing, the ability of the leading cloud providers to distribute software to match standardized APIs varies. Microsoft is in the software distribution business, so it would seem to be a small step for Azure to publish all the APIs and offer software to run in the datacenter. Interestingly for AWS and Google AppEngine, the best examples of standardizing APIs come from open sources efforts from outside these companies. Hadoop and Hypertable are efforts to recreate the Google infrastructure [11], and Eucalyptus recreates important aspects of the EC2 API [34].

[10]Indeed, harking back to Section 2, "surge chip fabrication" is one of the common uses of "chip-les" fabrication companies like TSMC.

[11]A 1TB 3.5" disk weighs 1.4 pounds. If we assume that packaging material adds about 20% to the weight, the shipping weight of 10 disks is 17 pounds. FedEx charges about $100 to deliver such a package by 10:30 AM the next day and about $50 to deliver it in 2 days. Similar to Netflix, Amazon might let you have one "disk boat" on loan to use when you need it. Thus, the round-trip shipping cost for Amazon to ship you a set of disks and for you to ship it back is $150, assuming 2-day delivery from Amazon and overnight delivery to send it to Amazon. It would then take Amazon about 2.4 hours to "dump" the disk contents into their datacenter (a 1 TB disk can transfer at 115 Mbytes/sec). If each disk contains whole files (e.g. a Linux ext3 or Windows NTFS filesystem), all disks could be read or written in parallel. While it's hard to put a cost of internal data center LAN bandwidth, it is surely at least 100x less expensive than WAN bandwidth. Let's assume the labor costs to unpack disks, load them so that they can be read, repackage them, and so on is $20 per disk.

The total latency is then less than a day (2.4 hours to write, 14-18 hours for overnight shipping, 2.4 hours to read) at a cost about $400 ($50 to receive from Amazon, $100 to send to Amazon, $200 for labor costs, and $40 charge for internal Amazon LAN bandwidth and labor charges).

Rather than ship disks, another option would be to ship a whole disk array including sheet metal, fans, power suppliers, and network interfaces. The extra components would increase the shipping weight, but it would simplify connection of storage to the Cloud and to the local device and reduce labor. Note that you would want a lot more network bandwidth than is typically provided in conventional disk arrays, since you don't want to stretch the time load or unload the data.

[12] The relatively new company Data Domain uses specialized compression algorithms tailored to incremental backups, they can reduce the size of these backups by a factor of 20. Note that compression can also reduce the cost to utility computing providers of their standard storage products. Lossless compression can reduce data demands by factors for two to three for many data types, and much higher for some. The Cloud Computing

provider likely has spare computation cycles at many times that could be used to compress data that has not been used recently. Thus, the actual storage costs could be two to three times less than customers believe they are storing. Although customers could do compression as well, they have to pay the computing cycles to compress and decompress the data, and do not have the luxury of "free" computation.

A second advantage that customers cannot have is to "de-dupe" files across multiple customers. This approach to storage calculates a signature for each file and then only stores a single copy of it on disk. Examples of files that could be identical across many customers include binaries for popular programs and popular images and other media.

[13]For example, to simplify parallel programming its common to have phases where all the threads compute and then all the threads communicate. Computation and communication phases are separated by a barrier synchronization, where every thread must wait until the last thread is finishing computing or communicating. If some threads of the gang are not running, that slows down these phases until they all have run. Although we could ask high-performance computing programmers to rewrite their programs using more relaxed synchronization, such as that found in Google's Map Reduce, a shorter term option would just be for the Cloud Computing provider to offer simultaneous gang scheduling of virtual machines as a Utility Computing option.

[14]Among Amazon's earliest offering was S3, a primary-key-only store for large binary objects. While S3 manages its own replication, failure masking and provisioning, the programmatic API is that of a key-value store (i.e., a hash table), the response time is not adequate for interactive client-server applications, and the data stored in S3 is opaque from the storage system's point of view (i.e., one cannot query or manage data based on any property other than its arbitrary primary key). Amazon's Elastic Block Store service allows customers to create a file system on a virtualized block device, but resource management and long-term redundancy are left to the programmer of each application; this represents an "impedance mismatch" with application developers, who now routinely rely on storage systems that perform additional resource management and provide an API that exposes the structure of the stored data. Amazon S3 and Google BigTable do this automatically, but their programmatic APIs do not expose much of the structure of the stored data, in contrast to relational databases such as Amazon SimpleDB or Microsoft SQL Data Services.

[15]The AWS announcement of Oracle product licensing only applies to users who are already Oracle customers on their local computers.

References

[1] Cloudera, Hadoop training and support [online]. Available from: `http://www.cloudera.com/`.

[2] TC3 Health Case Study: Amazon Web Services [online]. Available from: `http://aws.amazon.com/solutions/case-studies/tc3-health/`.

[3] Washington Post Case Study: Amazon Web Services [online]. Available from: `http://aws.amazon.com/solutions/case-studies/washington-post/`.

[4] Amazon.com CEO Jeff Bezos on Animoto [online]. April 2008. Available from: `http://blog.animoto.com/2008/04/21/amazon-ceo-jeff-bezos-on-animoto/`.

[5] Black Friday traffic takes down Sears.com. *Associated Press* (November 2008).

[6] ABRAMSON, D., BUYYA, R., AND GIDDY, J. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems 18*, 8 (2002), 1061–1074.

[7] ADMINISTRATION, E. I. State Electricity Prices, 2006 [online]. Available from: `http://www.eia.doe.gov/neic/rankings/stateelectricityprice.htm`.

[8] AMAZON AWS. Public Data Sets on AWS [online]. 2008. Available from: `http://aws.amazon.com/publicdatasets/`.

[9] BARROSO, L. A., AND HOLZLE, U. The Case for Energy-Proportional Computing. *IEEE Computer 40*, 12 (December 2007).

[10] BECHTOLSHEIM, A. Cloud Computing and Cloud Networking. talk at UC Berkeley, December 2008.

[11] BIALECKI, A., CAFARELLA, M., CUTTING, D., AND O'MALLEY, O. Hadoop: a framework for running applications on large clusters built of commodity hardware. *Wiki at http://lucene. apache. org/hadoop*.

[12] BRODKIN, J. Loss of customer data spurs closure of online storage service 'The Linkup'. *Network World* (August 2008).

[13] CARR, N. Rough Type [online]. 2008. Available from: `http://www.roughtype.com`.

[14] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)* (2006).

[15] CHENG, D. PaaS-onomics: A CIO's Guide to using Platform-as-a-Service to Lower Costs of Application Initiatives While Improving the Business Value of IT. Tech. rep., LongJump, 2008.

[16] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation* (Berkeley, CA, USA, 2004), USENIX Association, pp. 10–10.

[17] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (2007), ACM Press New York, NY, USA, pp. 205–220.

[18] DEMERS, A. J., PETERSEN, K., SPREITZER, M. J., TERRY, D. B., THEIMER, M. M., AND WELCH, B. B. The bayou architecture: Support for data sharing among mobile users. In *Proceedings IEEE Workshop on Mobile Computing Systems & Applications* (Santa Cruz, California, August-September 1994), pp. 2–7.

[19] GARFINKEL, S. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS . Tech. Rep. TR-08-07, Harvard University, August 2007.

[20] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM, pp. 29–43. Available from: http://portal.acm.org/ft_gateway.cfm?id=945450&type=pdf&coll=Portal&dl=GUIDE&CFID=19219697&CFTOKEN=50259492.

[21] GRAY, J. Distributed Computing Economics. *Queue 6*, 3 (2008), 63–68. Available from: http://portal.acm.org/ft_gateway.cfm?id=1394131&type=digital%20edition&coll=Portal&dl=GUIDE&CFID=19219697&CFTOKEN=50259492.

[22] GRAY, J., AND PATTERSON, D. A conversation with Jim Gray. *ACM Queue 1*, 4 (2003), 8–17.

[23] HAMILTON, J. Cost of Power in Large-Scale Data Centers [online]. November 2008. Available from: http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx.

[24] HAMILTON, J. Internet-Scale Service Efficiency. In *Large-Scale Distributed Systems and Middleware (LADIS) Workshop* (September 2008).

[25] HAMILTON, J. Perspectives [online]. 2008. Available from: http://perspectives.mvdirona.com.

[26] HAMILTON, J. Cooperative Expendable Micro-Slice Servers (CEMS):Low Cost, Low Power Servers for Internet-Scale Services. In *Conference on Innovative Data Systems Research (CIDR '09)* (January 2009).

[27] HÖLZLE, U. Private communication, January 2009.

[28] HOSANAGAR, K., KRISHNAN, R., SMITH, M., AND CHUANG, J. Optimal pricing of content delivery network (CDN) services. In *The 37th Annual Hawaii International Conference onSystem Sciences* (2004), pp. 205–214.

[29] JACKSON, T. We feel your pain, and we're sorry [online]. August 2008. Available from: http://gmailblog.blogspot.com/2008/08/we-feel-your-pain-and-were-sorry.html.

[30] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected operation in the coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles* (Asilomar Conference Center, Pacific Grove, U.S., 1991), vol. 25, ACM Press, pp. 213–225.

[31] KREBS, B. Amazon: Hey Spammers, Get Off My Cloud! *Washington Post* (July 2008).

[32] MCCALPIN, J. Memory bandwidth and machine balance in current high performance computers. *IEEE Technical Committee on Computer Architecture Newsletter* (1995), 19–25.

[33] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., , AND TURNER, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (April 2008).

[34] NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems . Tech. Rep. 2008-10, University of California, Santa Barbara, October 2008.

[35] PARKHILL, D. *The Challenge of the Computer Utility*. Addison-Wesley Educational Publishers Inc., US, 1966.

[36] PAXSON, V. private communication, December 2008.

[37] RANGAN, K. The Cloud Wars: $100+ billion at stake. Tech. rep., Merrill Lynch, May 2008.

[38] SIEGELE, L. Let It Rise: A Special Report on Corporate IT. *The Economist* (October 2008).

[39] STERN, A. Update From Amazon Regarding Friday's S3 Downtime. *CenterNetworks* (February 2008). Available from: http://www.centernetworks.com/amazon-s3-downtime-update.

[40] STUER, G., VANMECHELEN, K., AND BROECKHOVE, J. A commodity market algorithm for pricing substitutable Grid resources. *Future Generation Computer Systems 23*, 5 (2007), 688–701.

[41] THE AMAZON S3 TEAM. Amazon S3 Availability Event: July 20, 2008 [online]. July 2008. Available from: http://status.aws.amazon.com/s3-20080720.html.

[42] VOGELS, W. A Head in the Clouds—The Power of Infrastructure as a Service. In *First workshop on Cloud Computing and in Applications (CCA '08)* (October 2008).

[43] WILSON, S. AppEngine Outage. *CIO Weblog* (June 2008). Available from: http://www.cio-weblog.com/50226711/appengine_outage.php.

Great Principles of Computing

The great principles of computing have been interred beneath layers of technology in our understanding and our teaching. It is time to set them free.

Computer science was born in the mid-1940s with the construction of the first electronic computers. In just 60 years, computing has come to occupy a central place in science, engineering, business, and everyday life. Many whose lives are touched by computing want to know how computers work and how dangerous or risky they are; some want to make a profession from working with computers; and most everyone asks for an uncomplicated framework for understanding this complex field. Can their questions be answered in a compact, compelling, and coherent way?

In what follows, I will answer affirmatively, offering a picture of the great principles of computing. There are two kinds: principles of computation structure and behavior, which I call mechanics, and principles of design. What we call principles are almost always distilled from recurrent patterns observed in practice. Do practices shape to underlying principles? Do principles shape to practice? It is impossible to tell. In my description, therefore, I portray principles and practices as two equal dimensions of computing.

A principles-based approach is not new to science. The mature disciplines such as physics, biology, and astronomy portray themselves with such an approach. Each builds rich structures from a small set of great principles. Examples of this approach are *Lectures in Physics* by Richard Feynman [4], *The Joy of Science* by Robert Hazen and James Trefil [5], and *Cosmos* by Carl Sagan [7]. Newcomers find a principles-based approach to be much more rewarding because it promotes understanding from the beginning and shows how the science transcends particular technologies.

In my portrait, the contexts of use and their histories are imbued into principles, computing practices, and core technologies. Indeed, you cannot understand a principle without knowing where it came from, why it is important, why it is recurrent, why it is universal, and why it is unavoidable. Numerous application domains have influenced the design of all our core technologies. For example, the different styles of the languages Ada, Algol, Cobol, C++, Fortran, HTML, Java, Lisp, Perl, Prolog, and SQL flow out of the application domains that inspired them. You cannot make sense of the debates about the limits of machine intelligence without understanding cognitive science and linguistic philosophy. In software, unless you understand the

MICHAEL SCHROETER

The Profession of IT

The principles of a field are actually a set of interwoven stories about the structure and behavior of field elements.

different ways engineers and architects use the term "design," you cannot make sense of the tug-of-war between traditionalists promoting systems produced by a highly methodical engineering process, and agile developers promoting systems built for customer satisfaction, artistry, good taste, simplicity, and elegance.

The following sections describe principles of mechanics, principles of design, and computing practices as a framework supporting core technologies and application domains. A few implications of the framework on organization and content of computing curricula, and on the profession itself, are discussed at the end of this column.

Mechanics

In the 1950s, our field's founders portrayed their young science as a set of core technologies that supported application domains. They listed their core technologies as algorithms, numerical methods, computation models, compilers, languages, and logic circuits. Over the next 30 years, we added a few more: operating systems, information retrieval, databases, networks, artificial intelligence, human-computer interaction, and software engineering. The 1989 ACM/IEEE report, *Computing as a Discipline*, listed nine core technology areas [2]. Since then, the total number of core technology areas has tripled (see Table 1). Today, learning the mechanics of these technologies and their hundreds of possible direct interactions has become a daunting challenge.

algorithms	management information systems
artificial intelligence	natural-language processing
compilers	networks
computational science	operating systems
computer architecture	parallel computation
data mining	programming languages
data security	real-time systems
data structures	robots
databases	scientific computation
decision support systems	software engineering
distributed computation	supercomputers
e-commerce	virtual reality
graphics	vision
human-computer interaction	visualization
information retrieval	workflow

Table 1. Core technologies of computing.

In an effort to stem "curriculum bloat" from this growth, the *Curriculum 2001* report emphasizes the ideas at the intersection of the core technologies [3]. Unfortunately, a list of core technologies and their great ideas does little to convey the great principles of computing. Two books seeking to popularize computing focus on a few "great ideas" in a few of these areas, but their coverage is far from complete [1, 6]. Neither of these authors discusses which ideas are fundamental principles of all the core technologies.

Locating the fundamental principles of the field looks, therefore, to be a very attractive project. It calls to mind a picture in which the principles are the foundation of a pantheon with one pillar for each great principle. Unfortunately, as we shall soon see, such a picture is an unsatisfactory portrayal of computing.

Our initial question is: How shall we express our principles? It seems like we are looking for declarative statements, such as:

"The Turing machine is a model of universal computation."

"All information can be encoded as strings of bits."

"The number of bits in a message source is given by its entropy."

But this approach quickly becomes contentious. Some people argue over the definitions of terms like computation, informa-

tion, or message sources. Others ask whether some of the words ought to be qualified—such as algorithmic computation, physically represented bits, or discrete message sources. Still others ask why these statements are singled out and not others, such as "Every function imposes a lower-bound running time on all algorithms that compute it." Most everyone demands statements of obvious relevance to the familiar core technologies. But they wrestle over the selection criteria for principle statements, such as universality, recurrence, invariance, utility for prediction, or scope of consequences.

How do other fields express their principles? Physicists use terms like photons, electrons, quarks, quantum wave function, relativity, and energy conservation. Astronomers use terms like planets, stars, galaxies, Hubble shift, and black holes. Thermodynamicists uses terms like entropy, first law, second law, and Carnot cycle. Biologists use terms like phylogeny, ontogeny, DNA, and enzymes. Each of these terms is actually *the title of a story!* The principles of a field are actually a set of interwoven stories about the structure and behavior of field elements. They are the names of chapters in books about the field [4, 5, 7].

These principle-stories seek to make simple the complex history of a complex area. They tell history, showing how the principle evolved and grew in acceptance over time. They name the main contributors. They chronicle feats of heroes and failures of knaves. They lay out obstructions and how they were overcome. They explain how the principle works and how it affects everything else. The game is to define many terms in terms of a few terms and to logically derive many statements from a few statements.

Window	Central Concern	Principal Stories
Computation	What can be computed; limits of computing	Algorithm; control structures; data structures; automata; languages; Turing machines; universal computers; Turing complexity; Chaitin complexity; self-reference; predicate logic; approximations; heuristics; non-computability; translations; physical realizations
Communication	Sending messages from one point to another	Data transmission; Shannon entropy; encoding to medium; channel capacity; noise suppression; file compression; cryptography; reconfigurable packet networks; end-to-end error checking
Coordination	Multiple entities cooperating toward a single result	Human-to-human (action loops; workflows as supported by communicating computers); human-computer (interface; input; output; response time); computer-computer (synchronizations; races; deadlock; serializability; atomic actions)
Automation	Performing cognitive tasks by computer	Simulation of cognitive tasks; philosophical distinctions about automation; expertise and expert systems; enhancement of intelligence; Turing tests; machine learning and recognition; bionics
Recollection	Storing and retrieving information	Hierarchies of storage; locality of reference; caching; address space and mapping; naming; sharing; thrashing; searching; retrieval by name; retrieval by content

Table 2. The five windows of computing mechanics.

Astronomy, thermodynamics, and physics use the term *mechanics* for the part of their fields dealing with the behavior and structure of components. For example, Celestial Mechanics deals with the motions of heavenly bodies; Statistical Mechanics with the macro behavior of physical systems comprising large numbers of small particles; Quantum Mechanics with wave behaviors of subatomic particles; Rigid-Body Mechanics with the balance of forces within and between connected objects. I adopt this term for computing.

Computing Mechanics deals with the structure and operation of computations. It does so with stories for algorithm, Turing machine, grammar, message entropy, process, protocol stack, naming, caching, machine learning, virtual machine, and more. I found I could group the stories into the five categories of computation, communication, coordination, automation, and recollection (see Table 2). Every core technology expresses all five in its own way.

The lines between these categories are blurry. For example, the Internet protocol stack is an element of both communication and

The Profession of IT

coordination; naming and caching are both elements of communication and recollection. Therefore, I found it better to view the categories as windows into computing mechanics (see Figure 1). Although the views through the edges of windows overlap, the view through the centers is distinctive.

Design

Computing Mechanics does not exhaust all the principles of our field. Computing professionals follow principles of design that enable them to harness mechanics in the service of users and customers. Five concerns drive the design principles:

- Simplicity: Various forms of abstraction and structure that overcome the apparent complexity of the applications.
- Performance: predicting throughput, response time, bottlenecks, capacity planning.
- Reliability: redundancy, recovery, checkpoint, integrity, system trust.
- Evolvability: adapting to changes in function and scale.
- Security: access control, secrecy, privacy, authentication, integrity, safety.

The design principles themselves include abstraction, information hiding, modules, separate compilation, packages, version control, divide-and-conquer, functional levels, layering, hierarchy, separation of concerns, reuse, encapsulation, interfaces, and virtual machines. These principles are *conventions* that we collectively have found to lead consistently to dependable and useful programs, systems, and applications. These conventions are practiced within constraints of cost, schedule, compatibility, and usability.

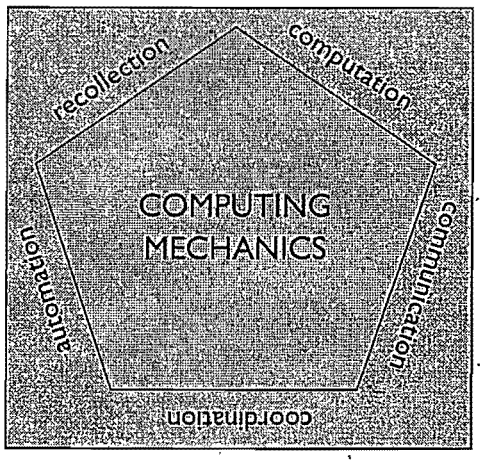


Figure 1. The five windows.

Design is not the same in computing as it is in other fields. In computing we design *abstract objects* that *perform actions*. Other fields use abstraction to explain or to organize tangible objects. Since design tells us about arrangements of basic components, design sits above mechanics in our picture of the field.

Might we call Computing Mechanics the "science" of computing and the Design Principles the "art"? I think not. There is good science and engineering and much art at all levels—mechanics, design, and applications.

Computing Practices

Our picture of computing needs more than mechanics and design. It needs an account of the computing practices that characterize our skills as professionals. Our competence is judged not by our ability to explain principles, but by the quality of what we do. I found five main categories of computing practice:

- Programming: Using programming languages to build software systems that meet specifications created in cooperation with the users of those systems. Computing professionals must be multilingual, facile with the numerous programming languages, each attuned to its own strategies for solving problems.
- Engineering Systems: Designing and constructing systems of software and hardware components running on servers connected by networks. These practices include a design component concerned with organizing a system to produce valuable and tangible benefits for the users; an engineering component concerned with the modules, abstractions, revisions, design decisions, and risks in the system; and an operations component concerned with configuration, management, and maintenance of the system. High levels of skill are needed for large programmed systems encompassing thousands of modules and millions of lines of code.
- Modeling and validation: Building models of systems to make predictions about their behavior under various conditions;

Our competence is judged not by our principles, but by the quality of what we do.

and designing experiments to validate algorithms and systems.

• Innovating: Exercising leadership to design and bring about lasting changes to the ways groups and communities operate. Innovators watch for and analyze opportunities, listen to customers, formulate offers customers see as valuable, and manage commitments to deliver the promised results. Innovators are history-makers who have strong historical sensibilities.

• Applying: Working with practitioners in application domains to produce computing systems that support their work. Working with other computing professionals to produce core technologies that support many applications.

I cannot overemphasize the importance of including computing practices in a portrait of our field. If we adopt a picture that ignores practices, our field will end up like the failed "new math" of the 1960s—all concepts, no practice, lifeless; dead.

Our portrait is now complete (see Figure 2). It consists of computing mechanics (the laws and universal recurrences that govern the operation of computations), design principles (the conventions for designing computations), computing practices (the standard ways of building and deploying computing systems), and core technologies (organized around shared attributes of application domains). Although not shown in the figure, the entire framework floats in a rich contextual sea of application domains, collectively exerting strong influences on core technologies, design, mechanics, and practice. Each level of the picture has a characteristic question that justifies its place in the hierarchy and exposes the integral role of practices (see Table 3).

Levels	Central Questions	Example Technologies
Application Domains	How do we work with others to design computing that serves them?	Supercomputers, grid computing, domain databases, graphics, design, interfaces, ...
Core Technologies	How do we design computations that support common elements across applications?	Algorithms, databases, networks, operating systems, HCI, AI, ...
Design Principles	How do we organize ourselves to build computations that work?	Design tools, object-oriented programming, layering, virtual machines, authentication, ...
Computing Mechanics	How do computations work?	Logic simulators, protocol stack, workflow, expert systems, virtual memory, ...

Table 3. Levels of action in computing practices.

Implications

By aligning with traditions of other science fields, a portrait of computing organized around great principles and practices promotes greater understanding of the science and engineering behind information technology. It significantly improves our ability to discuss risks, benefits, capabilities, and limitations with people outside the field. It recognizes that computing is action-oriented and has many customers, and that the context in which computing is used is as important as the mechanics of computing. It also clarifies professional competence, which depends on dexterity with mechanics, design, practices, core technologies, and applications.

For years, many others have seen our field as programming. Through our 1989 *Computing as a Discipline* report [3] we hoped to encourage new curricula that would overcome this misleading image. But this was not to be. Our practice of embedding a programming language in the first courses, started when languages were easy for beginners, has created a monster. Our students are being overwhelmed by the complexities of languages that many experts find

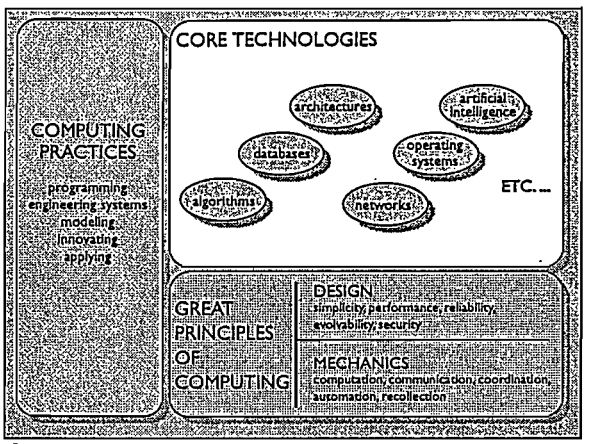


Figure 2. Principles-based portrait of computing.

challenging (typically Java and C++). Many students have turned to cheating and plagiarism as ways to pass these courses, and 35%–50% drop out prematurely. Many do not experience the joy of computing: the interplay between the great principles, the ways of algorithmic thinking, and the solutions of interesting problems. A curriculum organized around the framework offered here may rescue us from this unfortunate predicament. The current first courses (CS1, CS2, ...) can be replaced by Computing Mechanics (CM1, CM2, ...) and their extensive programming content can be moved to Programming Practices courses (PP1, PP2, ...) embedded within a larger Computing Practices track. The standard core courses (for example, algorithms, operating systems, databases, software engineering, or networks) can then be reshaped to extend computing mechanics into their areas rather than teaching applicable mechanics from scratch. (Aside to academic colleagues: starting with computing mechanics is not a "breadth-first" approach; the framework promotes depth in concepts, design, and practice.)

It is time for us to make ourselves known by saying our mechanics, our design principles, and our practices. It is time to stop hiding the enormous depth and breadth of our field.

References

1. Biermann, A. *Great Ideas in Computer Science, 2nd ed.* MIT Press, 1997.
2. Denning, P., et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
3. *Curriculum 2001 Final Report*; computer.org/education/cc2001/final/.
4. Feynman, R. *Lectures in Physics.* Addison-Wesley, 1970.
5. Hazen, R. and Trefil, J. *Science Matters.* Anchor, 1991.
6. Hillis, D. *The Pattern on the Stone.* Basic Books, 1999.
7. Sagan, C. *Cosmos.* Random House, 2002.

PETER DENNING (pjd@nps.navy.mil) is past president of ACM.