Axiomatic Cloud Theory

Working Paper, July 29, 2011

Joe Weinman[1]

Permalink: http://www.JoeWeinman.com/Resources/Joe_Weinman_Axiomatic_Cloud_Theory.pdf

Abstract

There is a long tradition of formal models of systems and processes for computation—sequential and parallel, deterministic and non-deterministic—for example, finite state automata, Turing machines, and Petri nets. Such abstract models can help characterize and thus provide insight into the behavior of real-world systems.

Today, cloud computing has become of great interest technically and as it relates to business strategy and competitiveness. However, while there are numerous informal (verbal) definitions of cloud computing, rigorous axiomatic, formal (mathematical) models of clouds appear to be in short supply: this paper is the first use of the term "Axiomatic Cloud Theory."

Herein we propose a formal model of the cloud, suitable for computing but also applicable to other domains where dispersed resources are dynamically allocated to customers under a usage-sensitive pricing scheme: car rentals, airlines, taxis, electric utilities, bank loans, etc.

Specifically, we propose mathematically precise cloud system and process definitions and cloud axioms, demonstrate a broad range of applications, derive some results, and comment on business implications of the results. This rigorous model draws on mature disciplines: set theory, metric spaces, measure theory, σ-algebras, graph theory, function spaces, linear algebra, etc. This formalizes what may well be one of the most well known definitions of cloud computing from the National Institute of Standards and Technology.

We define a cloud as a structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ satisfying five formal axioms: it must be 1) Common, 2) Location-independent, 3) Online, 4) Utility, and 5) on-Demand. $\mathbb{S}$ is space, $\mathbb{T}$ is time; $G = (V, E)$ is a directed graph; Q is a set of states, where each state combines assignments of resource capacity and demand, resource allocations, node location, and pricing; q_0 is an initial state; and δ is a transition function that determines state trajectories over time: mapping resources, allocations, locations, and pricing to a next state of resources, allocations, locations, and pricing. This captures the interrelationships in a real cloud: capacity relative to demand can drive pricing, pricing and resource location drive allocation, allocation patterns can drive new resource levels.

This approach may be viewed as a first step towards formal modeling of clouds. There will no doubt be those who wish to apply, restrict, extend, or modify the proposed model. It is to be hoped that this note initiates a constructive dialogue on formal models of cloud computing.

---

[1] Joe Weinman leads Communications, Media and Entertainment Industry Solutions for Hewlett-Packard. The views expressed herein are his own. Contact information is at http://www.joeweinman.com/contact.htm

1. Introduction

Cloud computing has become one of the top priorities for CIO's, and can have many benefits, including cost reduction, revenue growth, increased business agility, and enhanced customer experience. There are perhaps as many definitions of "cloud computing" as there are practitioners, commercial firms, institutions, and analysts in the field: one recent survey[2] reviewed 22 different definitions; no doubt there are many more. Many of the definitions overlap, however, suggesting that there is potential for a common view. One widely accepted definition that captures much of the essence of cloud computing has been refined by the U.S. National Institute of Standards and Technology:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[3]

We use this and related definitions as a basis to propose here a formal model of the cloud. Such a formal model, in the spirit of the axiomatic theory of geometry delineated in Euclid's *Elements*, Set Theory, Group Theory, Petri Net Theory, and many other fields, includes a set of mathematically precise definitions, axioms, and theorems. In addition, we illustrate examples of application of the model, and comment on the business implications of these theorems. The purpose of this note, however, is not to exhaustively derive the theory, but rather to argue—by proposing a model and attempting to demonstrate its value—that an axiomatic foundation for the cloud merits investigation and refinement.

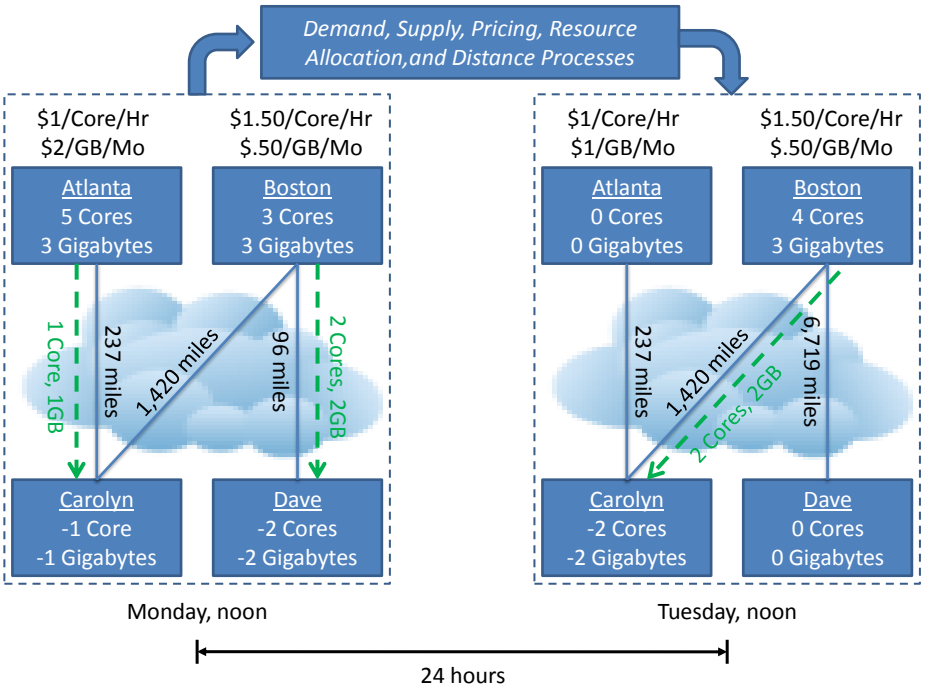
Such an endeavor is something of a balancing act. It must be general enough to be applicable to a range of conditions, but specific enough not to be a theory of everything. It must be relevant to cloud computing, but represent the truths about the behavior of structures and systems that resemble computing clouds, but are outside of the domain of computing. It must be mathematically rigorous, without being unmanageable for the general reader. Moreover, no doubt there will be readers that will agree with some definitions and axioms, disagree with others, wish to change some or suggest new ones. As with Euclid's 5th Postulate, such considerations led to spherical, elliptic, and hyperbolic geometry. Moreover, increasing rigor in the underlying mathematics meant refinements to the axiomatic foundation of geometry including work by Hilbert, Birkhoff, and Tarski, and in set theory, Russell's Paradox based on the Axiom of Specification led to the Zermelo-Fraenkel axiomatization.

---

[2] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, Maik Lindner, "A Break in the Clouds: Towards a Cloud Definition," ACM SIGCOMM Computer Communications Review, Volume 39, Number 1, January, 2009.

[3] Peter Mell and Tim Grance, "The NIST Definition of Cloud Computing," v. 15, NIST Special Publication 800-145 (Draft), January 2011, at http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf.

Informally, we are modeling an environment that, for example, might look like this:



In words: there is a set of four nodes or vertices: $\{Atlanta, Boston, Carolyn, Dave\}$. They are connected by a network of three edges: $\{Atlanta - Carolyn, Boston - Carolyn, and\ Boston - Dave\}$ with associated distances as shown. On Monday at noon, $Atlanta$ and $Boston$ both have resources, e.g., $Atlanta$ *has* 5 Cores and 3 GB, which we can denote as $< 5,3 >$. At the same time, $Carolyn$ *needs* 1 Core and 1 GB, which we can denote as a capacity of $< -1, -1 >$. 1 Core and 1 GB has been allocated to $Carolyn$ from $Atlanta$ by a resource allocation process. The price for these resources is one dollar per core per hour, and two dollars per GB per month. Twenty four hours later, conditions have changed: $Carolyn$ needs more resources, $Dave$ needs fewer. Pricing has changed, perhaps to stimulate demand. Resource availability has been changed, possibly because $Atlanta$ is conducting scheduled maintenance or has had an unplanned outage.

Generally, there will be a variety of interacting processes. Due to the lack of resources in $Atlanta$, $Carolyn$ has been allocated resources from $Boston$. $Dave$ has left his home in the Boston suburbs and taken a transpacific flight, and so his distance to Boston has increased by thousands of miles. Note that many different interactions may result in this state change: demand processes may demonstrate price elasticity of demand; yield management may lead to dynamic pricing; shifts in demand, pricing, distances or resource availability may lead to shifts in how resources are allocated. Note that in this model we can clearly distinguish between suppliers ($Atlanta$, $Boston$), and customers ($Carolyn$, $Dave$). However, generally we may choose to have exchanges between market agents, or have some nodes act as intermediaries, distributors, aggregators, or transformation agents (e.g., manufacturers). Also, while this model shows a single price per supplier, pricing may be more general, e.g., a single market price, or

more specific, e.g., price discrimination where each price is based on a specific supplier-customer relationship.

Keeping this informal scenario in mind, we will define in the coming pages a cloud as a specific structure that satisfies five cloud axioms. Some formal background is assumed, which is overviewed in the appendix.

2. Cloud Structure and Axioms

We will now define the elements of the informal scenario addressed above more formally. A *cloud structure* is 6-tuple $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$. Later, we will couple this structure with 5 axioms to define a *cloud*. Briefly, $\mathbb{S}$ is space, $\mathbb{T}$ is time; $G = (V, E)$ is a directed graph; Q is a set of states combining resource capacity and demand, resource allocations, node location, and pricing; q_0 is an initial state; and δ is a transition function that maps a graph, resources, allocations, locations, and pricing to a next state of resources, allocations, locations, and pricing. One may recognize some aspects of finite state automata, Turing machines, Petri nets, and other constructs in this model. To be more precise,

Definition: a *cloud structure* is a 6-tuple $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$, where

Space $\mathbb{S} = (M, \lambda)$ is a metric space, where
- M is a set of *locations*, and
- λ is a distance metric, $\lambda: M \times M \to \mathbb{R}_0^+$;

Time $\mathbb{T} = (T, \Sigma, \tau, <)$ is a measure space with a strict total ordering and least element $t_0 \in T$, where
- T is a set of periods,
- Σ is a σ-algebra over T,
- $\tau: \Sigma \to \mathbb{R}_0^+$ is a measure on Σ, and
- $<$ is a strict total ordering on T;

Network $G = (V, E)$ is a simple, oriented graph, with no self-loops or multiple edges, where
- V is a set of vertices, and
- E is a set of edges, $E \subseteq V \times V$, and $(u, v) \in E$ implies $(v, v) \notin E$ and $(v, u) \notin E$;

$Q = \{q_0, q_1, q_2 \ldots\}$ is a set of *states,* where each $q_j = (R_{q_j}, A_{q_j}, L_{q_j}, P_{q_j})$, where
- $R_{q_j}: V \to \mathbb{R}^r$ is a *resource* function,
- $A_{q_j}: E \to \mathbb{R}^r$ is an *allocation* function,
- $L_{q_j}: V \to M$ is a *location* function, and
- $P_{q_j}: E \to F$ is a *pricing* function, where F is the function space $F = \{f | f: \mathbb{R}^r \times \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}\}$;

$\delta: T \to Q$ is a transition mapping; and

$q_0 \in Q$, where $q_0 = \delta(t_0)$.

This definition is formally rigorous, but perhaps not optimally comprehensible, so we expound further on the elements.

Space $\mathbb{S} = (\boldsymbol{M}, \boldsymbol{\lambda})$ is a metric space, where $\boldsymbol{M}$ is a set of *locations*, and $\boldsymbol{\lambda}$ is a distance metric, $\boldsymbol{\lambda}: \boldsymbol{M} \times \boldsymbol{M} \to \mathbb{R}_{\boldsymbol{0}}^{+}$;

Metric spaces are merely sets with non-negative distances defined between their elements.

One such space is 3-dimensional space $\mathbb{R}^3$, with a "Euclidean" metric defined on it, i.e., the distance λ between x and y where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$ is just $\lambda(x, y) = \sqrt{(\,y_1 - x_1\,)^2 + (\,y_2 - x_2\,)^2 + (\,y_3 - x_3\,)^2}$.

Another such space is a 2-dimensional plane $\mathbb{R}^2$ with a "Manhattan" metric defined on it, after the number of city blocks you'd have to walk (no diagonals allowed). This is just $\lambda(x, y) = |y_1 - x_1| + |y_2 - x_2|$.

Another metric space is the points of a sphere, $M = \{\mathbb{R}^3: \sqrt{r_1^2 + r_2^2 + r_3^2} = 1\}$, and λ is the shortest great circle route.

Generally, $S \subseteq \mathbb{R}^{\boldsymbol{q}}$, $q \geq 1$, but we may also have $S = \{"Boston," "Austin," "Bob's\ House"\}$

"Distance" is a dimensionless quantity as used here, but may represent miles, milliseconds of delay, router hops, or other characteristics.

Time $\mathbb{T} = (\boldsymbol{T}, \boldsymbol{\Sigma}, \boldsymbol{\tau}, <)$ is a measure space with a strict total ordering and least element $\boldsymbol{t_0} \in \boldsymbol{T}$, where $\boldsymbol{T}$ is a set of periods, $\boldsymbol{\Sigma}$ is a $\boldsymbol{\sigma}$-algebra over $\boldsymbol{T}$, $\boldsymbol{\tau}: \boldsymbol{\Sigma} \to \mathbb{R}_{\boldsymbol{0}}^{+}$ is a measure on $\boldsymbol{\Sigma}$, and $<$ is a strict total ordering on $\boldsymbol{T}$;

Measure spaces are formally defined constructs. For the purposes of this document, the only important thing is that T is a set of time periods, each of which may be infinitesimal instants, e.g., $T = [0,60]$ may correspond to the uncountably infinite number of "real" instants in a 60 second interval, or T may be abstract steps $\{t_0, t_1, t_2, t_3, \ldots\}$, or may be days {"Monday the 3rd," "Tuesday the 4th,"Wednesday the 5th," …}. Σ, which is beyond the scope of this paper, is basically a nonempty set of subsets of T closed under union and complement with respect to T, and importantly, τ is a measure on Σ, or to put it another way, is a measure on subsets of T. Let $T_1 \in \Sigma$, and let $T_1 = [27,42.3]$. Then $\tau(T_1)$ might be 15.3. Or, let $T_1 = [Monday, Wednesday] \cup \{lunchtime\ Thursday\}$, then $\tau(T_1)$ might be 73 (hours). As with λ, in this model τ is dimensionless (although we often

conceive of distance in, say, feet and time in, say, hours). "<" is a strict total order on T, i.e., a way of formalizing "before" and "after."

τ may be a *counting measure*, e.g., 3 time periods, or a *Lebesque measure*, such as duration, e.g., 5 hours. Normally, μ is the symbol for a measure but we use τ to avoid confusion with the symbol for statistical mean μ, which also happens to be the symbol for a Petri net marking.

Network $G = (V, E)$ is a simple, oriented graph, with no self-loops or multiple edges, where V is a set of vertices, and E is a set of edges, $E \subseteq V \times V$, and $(u, v) \in E$ implies $(v, v) \notin E$ and $(v, u) \notin E$;

The graph is "oriented," which essentially means directed rather than undirected. The subtle difference is that each edge is still a directed arc, but this does not impact connectedness and does not in any way limit the possible allocations of resources. Instead, it establishes an orientation for which way an allocation "flows." G is simple, i.e., no self-loops or multiple edges. Typically G is finite, but in some models we may allow G to be infinite.

If $(u, v) \in E$, then u is a *predecessor* of v and v is a *successor* of u. We will use the notation I_v (for inputs) to indicate the set of vertices that are predecessors of v, i.e., all u such that $(u, v) \in E$, and O_u (for outputs) to indicate the set of vertices that are successors of u, i.e., all v such that $(u, v) \in E$.

$Q = \{q_0, q_1, q_2 \ldots\}$ is a set of *states,* where each $q_j = (R_{q_j}, A_{q_j}, L_{q_j}, P_{q_j})$, where $R_{q_j}: V \to \mathbb{R}^r$ is a *resource* function, $A_{q_j}: E \to \mathbb{R}^r$ is an *allocation* function, $L_{q_j}: V \to M$ is a *location* function, and $P_{q_j}: E \to F$ is a *pricing* function, where F is the function space $F = \{f | f: \mathbb{R}^r \times \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}\}$;

Any $q \in Q$ is a composite state made up of a resource assignment, an allocation assignment, a location assignment, and a pricing function assignment.

A *resource* assignment specifies a specific resource vector for each vertex. Each resource vector is taken from $\mathbb{R}^r$, where each dimension characterizes a different resource. For example, <1,2,4> may stand for 1 database instance, 2 servers, and 4 storage arrays, or in a different context, 1 hotel room, 2 beds, 4 pool passes. Negative values indicate requirements, rather than capacity. For example, <-1,-2,-1> may indicate that I need 1 database instance, 2 servers and 1 storage array.

An *allocation* defines a "flow" along the oriented edge connecting two vertices. Typically a flow will occur from a vertex with a positive value to a vertex with a negative one, but this is not always the case.

An allocation is conceptually different than a network "flow," in that allocated resources need not move, and bidirectional (i.e., full duplex) allocations may occur along an edge, signifying bilateral trade. Moreover, unlike classic network flows where nodes are only

of value in terms of routing flows, in the cloud model proposed here all nodes may have additional resources or demands in any given period. In a classic network flow model, there is only one node (the *source*) with all capacity, and one other node (the *sink*) with all demand. Nevertheless, we will often use the term "flow."

A *location* assignment tells us "where" the vertex is among the locations M. From this, we can derive a distance along or length of an edge $e = (u, v)$, using the distance metric λ, specifically via $\lambda(L(u), L(v))$.

While often locations and thus distance are fixed, rather than a function of time, mobile users (and even mobile service delivery nodes) may cause locations for nodes to move, and therefore for location and distance to be time-dependent. This is true of mobile cellular users, where the customers move, but also of taxi services: where both customer nodes and supplier nodes move: picture someone walking down the street trying to catch one of the many cabs that is continuously in motion.

Finally, a particular *pricing* lets us calculate total prices, and thus make economically optimal allocation decisions. For a given state $q \in Q$, where $q = \delta(t),\ t \in T$, each edge $e \in E$ has a *pricing* associated with it, which may be "0", i.e., "free." Such a pricing is associated with the edge, not the node, because there may be "price discrimination," where a particular supplier charges different customers differently. Also, pricing may change over time: this is so-called "dynamic pricing." Due to this, each particular state q specifies a pricing function P_q that defines a correspondence between each edge and its pricing (which is a function, not a value), so the pricing function P_q is actually a mapping whose domain is edges and whose codomain is the function space consisting of all the possible functions that can be defined using allocations (r-dimensional real vectors), distances (non-negative reals), and times (also non-negative reals). (As an aside, this function space turns out to be a vector space as well, but a completely different vector space than $\mathbb{R}^r$). In effect, P_q is a pricing function function.

Many different pricings are conceivable: free, flat-rate, distance-sensitive, pay-per-use, two-part tariffs, block-declining, and so forth. Often, pricing will be based on the size and composition of the allocation. Consequently, a pricing might be of the form $p \cdot a \times \tau(t)$, where p is a price vector, a is the allocation on the edge, "$\cdot$" is the vector inner product, and $\tau(t)$ is the duration of the time period, as quantified by the measure τ. Clearly p and a need to be of the same dimension r. This type of pricing is often encountered: dollars per kilowatt-hour for electricity, dollars per room-night for hotels, dollars per car-day for rental cars, dollars per server-hour for computing. Another type of pricing might be linearly distance-sensitive, where the pricing is of the form $k\lambda$, where k is a constant and λ is the distance. There also may be combinations of these: the function space is closed under addition.

A *price* for an edge at a specific time results from evaluating the *pricing* for that edge at that time (and thus state) using the allocation of the edge at that time and distance between the incident vertices of the edge at that time. We will discuss this more later.

Often, in network problems, one tends to talk about "costs." We deliberately use "price" to connote a commercial focus rather than an internal cost-optimization focus.

We can aggregate prices over all times in T for a given edge, or over all edges in E at a given time, or for the entire cloud structure across all edges given the locations of the nodes and all allocations given the resourcing of the nodes.

The state is a combination of elements for two reasons. One, it takes several components to capture the richness of the cloud model and thus a particular state. Secondly, while the components may be independent, some or all may interrelate. For example, aggregate supply and demand may impact price. Price may impact addition of new capacity. Distance, i.e., relative location, may impact price, and thus the allocation of customers to suppliers. Although "allocation" sounds like something done by a management process, we may also think of it as an emergent result caused by a set of independent decisions made by each node acting as rational, self-interested agent.

$\delta: T \rightarrow Q$ is a transition mapping;

We could also state this as δ maps a particular time into a particular state: given any time, δ gives us a specific state. Our definition of the transition function departs somewhat from traditional Turing machine or Petri net definition, although it encompasses those models.

In the simplest case, we may define δ to be constant over time: $\forall t \in T, \delta(t_i) = q_0$.

Or we may define the elements of Q strictly as functions of time. For example, $T = [0,1000]$, and $R(t) = \sqrt{t}$.

Or, in accordance with traditional discrete dynamic models and automata, we can effect a discrete state transition scheme based on some recursive function f by letting

$$\delta(t_i) = \begin{cases} f\big(\delta(t_{i-1})\big), i > 0 \\ q_0, i = 0 \end{cases}$$

Generally, we can have an arbitrary function defining the trajectory of δ, typically based on a computable, or recursively enumerable, function. In other words, the rules defining a transition may be arbitrarily complex. They also may be nondeterministic (stochastic), and either memoryless (e.g., Markov processes) or nonmemoryless.

$q_0 \in Q$, where $q_0 = \delta(t_0)$.

In other words, q_0 is the initial state, which is the state at time t_0. This may be redundant, in that if the mapping δ is fully defined, we can determine q_0. However, if δ is recursively defined, as above, we will need to have a first q_0 specified to "prime the pump."

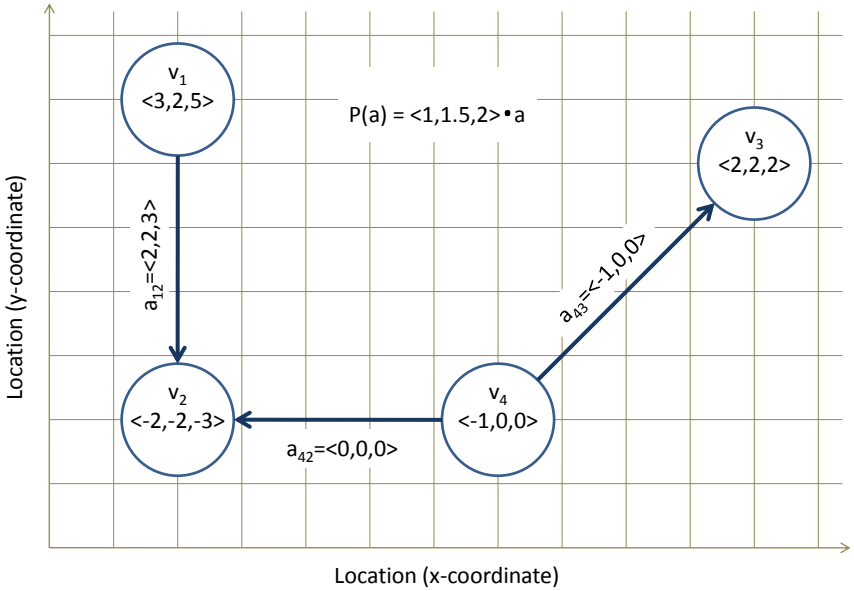
Note that, as defined, a cloud "structure" *encodes* execution. Since we have (perhaps overzealously) striven for rigor, we also need to consider readability and usability. Consequently, we will often use shorthand to describe various situations.

The transition mapping δ maps out a trajectory of the state space over time, beginning with q_0 at t_0. For any $t \in T$, we can determine a $q = \delta(t)$. Consequently, we can determine the components of q. We will use the shorthand of $R(t_i)$ to refer to R_{q_j}, where $q_j = \delta(t_i)$, and similarly for $A(t_i)$, $L(t_i)$, and $P(t_i)$. And we will refer to $R(t)$, $A(t)$, $L(t)$, and $P(t)$ when we wish to characterize the time-varying behavior of these functions, including when we characterize them by their distributions and statistics. Moreover since these are mappings from either vertices V of G or edges E of G, we can specifically refer to $R_v(t)$ or $L_v(t)$ when $v \in V$, or we can refer to $A_{uv}(t)$, $A_e(t)$, $P_{uv}(t)$ or $P_e(t)$ when $e = uv \in E$.

We will use λ_e as shorthand for $\lambda\big(L(u), L(v)\big)$, $uv = e \in E$.

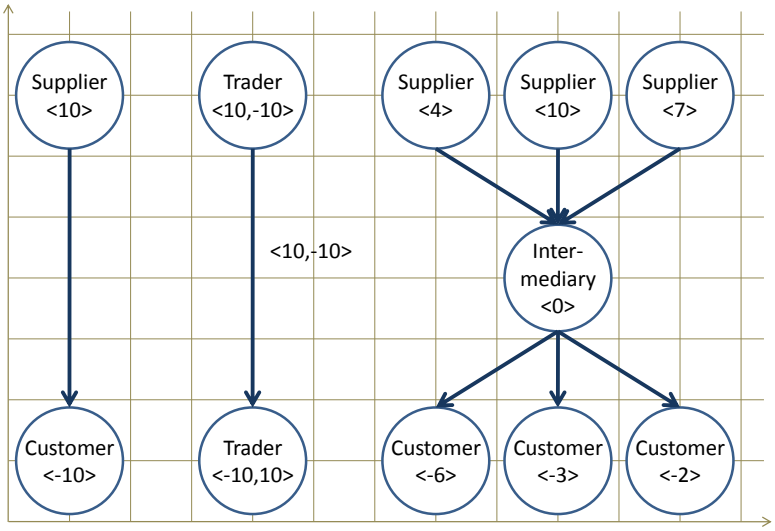
When resources are positive, generally we refer to the value as "capacity" and the node as a "supplier," when they are negative, we generally refer to the value as "demand" and the node as a "customer." We will sometimes write D_i or $D_i(t)$ as shorthand for $-R_i$ or $-R_i(t)$ respectively. When the vector includes positive and negative values, we may call the node a "trader."

This diagram helps illustrate many of the elements above.

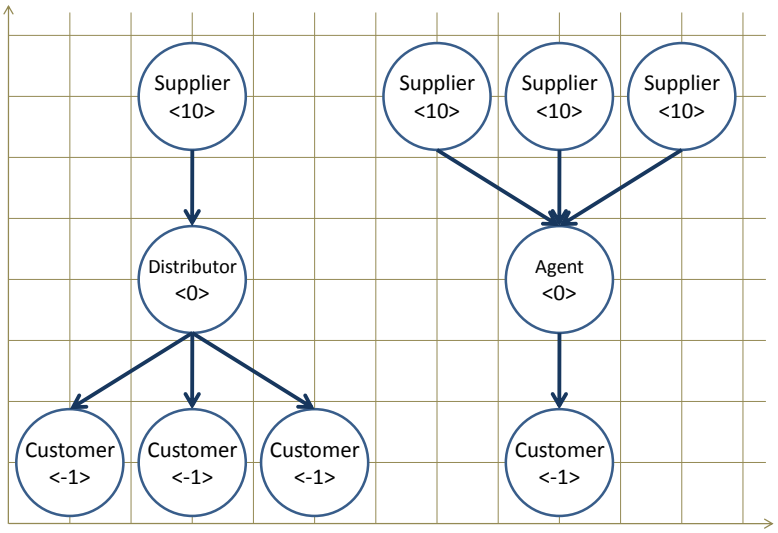


3. Examples

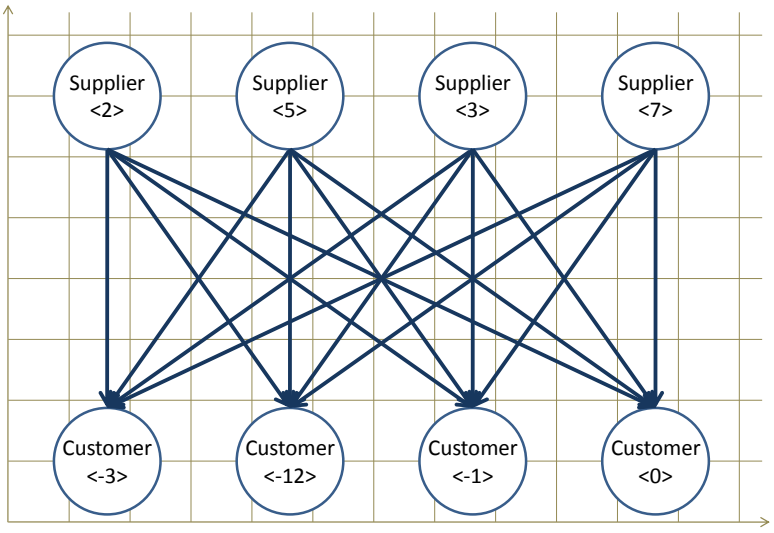
The cloud structure model can be used to model many different scenarios. The diagram below shows a straightforward supplier-customer model, where the supplier has resources, and the customer wants them. Or each may have something of value but want what the other has. This may be viewed as a "trade" or barter situation. Of course, one resource may be money, in which case a "sale" is actually an exchange. Or, both resources may be money, in which case this is a currency exchange. Recall that the direction of the arrow orients the resource flow, but doesn't constrain positive and negative resource flows in any way. Another case is an intermediary. One example might be a travel agent, where the suppliers are airlines and the customers are travelers.



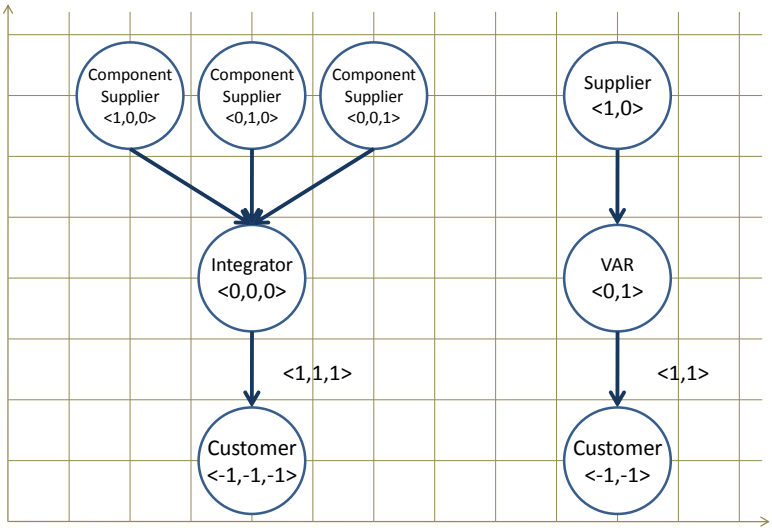
Variations of the intermediary model exist: distributors such as manufacturer's representatives, and buyer's agents:



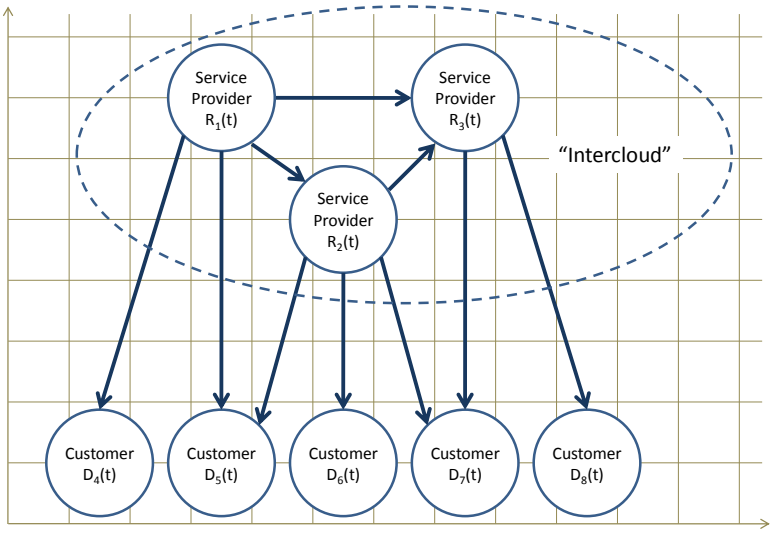
Sometimes, cloud structures can be clearly partitioned into a simple, bipartite structure, with clearly delineated suppliers with resources, clearly delineated customers with demands (i.e., negative resources), and always positive flows from suppliers to customers:



An integrator may take a different resource from each of multiple suppliers, and sell the assembly. A value-added reseller may take one product, service, or component, and add in "value," i.e., a different resource, and deliver the total to an end-customer.

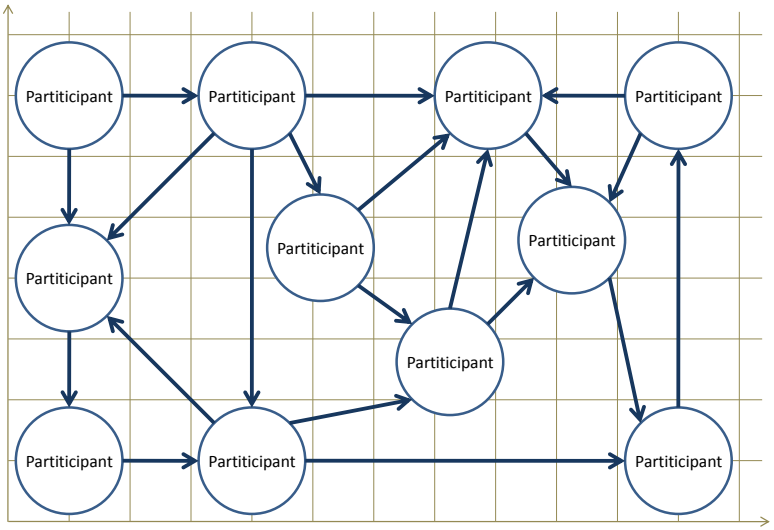


In the "intercloud" scenario, such as the electrical grid or the Reservoir[4] Cloud Computing project, there are multiple service providers, each of which services customers. However, if a provider has insufficient resources, it may acquire capacity temporarily from another provider.



Finally, there is the general case of a collaborative environment, firm, market or economy, with many participants interacting in many complex ways.

[4] B. Rochwerger, et al., "Reservoir—When One Cloud Is Not Enough," IEEE Computer, March 2011, pp. 44-51



4. Axioms

The notion of "axiom" is not as clear as one might wish. It sometimes means logical tautology, in the sense of an inarguable truth, common notion, or fundamental principle of logic. We use it here in the sense of a restriction, condition, or property, in the same way that, say, the *associativity* axiom for groups restricts the collection of *all* pairs of sets and operators to *only* those which are *associative*.

We stipulate 5 cloud axioms, (which conveniently spell "C.L.O.U.D.": Common, Location-independent, Online, Utility, and on-Demand. A cloud then, is a cloud structure that satisfies these five axioms. These may also be considered to be criteria, conditions, properties, restrictions, postulates, etc.

Following are some necessary definitions and the development of an axiomatic model for the cloud.

Definition: An allocation A is *feasible* at time t if and only if

$$\forall v \in V, \sum_{u \in I_v} A_{uv}(t) + R_v(t) - \sum_{x \in O_v} A_{vx}(t) \geq 0$$

To put it simply, some resources may not be allocated, but, in each dimension of $\mathbb{R}^r$, the sum of the inflows to a node together with the resources at that node must equal or exceed the sum of

the outflows from that node for the allocation to be feasible. For the special case of a feasible allocation where all the sums equal zero, we will call it *perfect*. If not feasible, we will call it *insufficient.*

Definition: A cloud structure is *completely feasible* if for all $t \in T$, $A(t)$ is feasible.

In other words, the structure is completely feasible if there is a feasible allocation at each time period.

Definition: Two edges $e, f \in E$ are *coincident* if they share a vertex, i.e., $\exists v \in V$, such that $v \in e$ and $v \in f$.

This is the traditional graph-theoretic definition of edge adjacency or coincidence.

Definition: Given $t \in T$, two nonzero flows $A_e(t)$ and $A_f(t)$ are *coincident in space* if $e, f \in E$ are coincident.

In other words, flows that are coincident in space exist on coincident edges.

Definition: Given $t_1, t_2 \in T, t_1 \neq t_2$, two nonzero flows $A_e(t_1)$ and $A_f(t_2)$ are *coincident over time* if $e, f \in E$ are coincident.

While such flows may not happen to be coincident at the same time, in other words.

For the next definition, let us assume that $u \in e$ and $u \in f$, and without loss of generality, let us assume that $e = uv$ and $f = ux$, i.e., that u is a predecessor of both v and x, since G is oriented. We can logically flip directions of arcs and take the inverse of the allocation without altering anything.

Definition: Two flows $A_e(t_1) =< a_{e_1}, a_{e_2}, \dots a_{e_r} >$ and $A_f(t_2) =< a_{f_1}, a_{f_2}, \dots a_{f_r} >$ on $e, f \in E$ respectively where $u \in e$ and $u \in f$ are *commonly sourced* if:

1) they are coincident in space or coincident over time; and

2) there is an $i, 1 \leq i \leq r$, such that either

a) $uv = e$, $uw = f$, $a_{e_i} > 0$, and $a_{f_i} > 0$

b) $vu = e$, $uw = f$, $a_{e_i} < 0$, and $a_{f_i} > 0$

c) $uv = e$, $wu = f$, $a_{e_i} > 0$, and $a_{f_i} < 0$

d) $vu = e$, $wu = f$, $a_{e_i} < 0$, and $a_{f_i} < 0$.

We are actually saying something very simple, but the four variations of which way the oriented edges may point makes it hard to describe. In effect, two flows are commonly sourced if there is a single node that is providing at least one positive resource to two other nodes, either at the same time, different times, or both.

We will define five key properties which will become axioms. Our first key definition:

Definition of "Common:" A cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *common* if and only if there are at least two nonzero flows that are commonly sourced.

This definition formalizes the notion of "resource sharing" or "common pools." Without the definition, *all* nodes would effectively be "paired off" into permanent, private or captive "customer-supplier" or "exchange" relationships, or linear chains.

Now let's turn to prices and pricing. This is a multilayered concept, which we will build up a step at a time.

Definition: Let the *edge price at time* t, $\hat{P}_e(t)$ be defined as the evaluation of the pricing function for that edge $e = uv \in E$ at that time, i.e., $\hat{P}_e(t) = f(A_e(t), \lambda(L(u), L(v)), \tau(t))$, where $f(\,) = P_q(e)$, i.e., the function that is the image of the edge e under P_q, where $q = \delta(t)$.

An edge price at time t, $\hat{P}_e(t)$ may be positive, in the traditional sense of a price, or negative, showing a net negative value for a set of positive and negative allocations (i.e., outbound and inbound trades) in a resource allocation vector.

An edge price at time t, $\hat{P}_e(t)$ may also be negative, for example, if we are modeling credits for SLA violations, or if there is a positive pricing function but the orientation of the edge is inverted relative to the allocation flow.

We can aggregate all the prices across all the edges at a given time:

Definition: We define the *total price at time* t, $\hat{P}(\mathrm{t})$ to be

$$\hat{P}(\mathrm{t}) = \sum_{\mathrm{e} \in \mathrm{E}} \hat{P}_e(t)$$

And finally, we can sum or integrate the total price across all of time:

Definition: The *total price* of a cloud structure is:

$$\hat{P} = \int_{t \in T} \hat{P}(\mathrm{t})$$

In the next definition, recall that B^A stands for the set of all (possible) mappings from A to B.

Recall that a state $q = (R_q, A_q, L_q, P_q)$. We can generate a related state by substituting one or more of the elements, e.g., $q' = (R_q, A', L_q, P_q)$.

Definition of "Location-independence:" A completely feasible cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *location-independent* if and only if $\forall t \in T$, if $q = \delta(t) = (R_q, A_q, L_q, P_q)$ and the total price at time t is $\hat{P}_q(\mathrm{t})$, then $\forall L' \in M^V$, $\exists A' \in \mathbb{R}^{rE}$, such that if $q' = (R_q, A', L', P_q)$, A' is feasible and $\hat{P}_{q'}(\mathrm{t}) \leq \hat{P}_q(\mathrm{t})$.

In other words, if location-independence holds, then if there is at least one feasible allocation at time t given a particular configuration of locations, then given *any other possible* location configuration, there *still* will exist some feasible allocation (which may or may not be the same one) that doesn't cost any more, and actually might cost less.

There is more than one way this can happen. One is that pricings for this cloud don't include distance as a parameter, and thus location doesn't matter. But another is that the topology of nodes with resources is sufficiently distributed and capacitated, and the distance function λ appropriately structured, such that we can "rehome" nodes with demand to get their flow from a sufficiently nearby node in such a way that the total price at each time is not negatively impacted.

Definition of "Online:" A cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *online* if and only if G is weakly connected, i.e., for $\forall u, v \in V$, there is a semipath from u to v.

"Weakly connected" means that, ignoring the direction (or orientation) of each edge, there is always a path between any two vertices. More formally, this is called a "semipath,"[5] as opposed to a directed path, where we can follow the direction of the arrows.

Simply put, there are no isolated customers, suppliers, or traders, nor are there isolated components.

Definition: A pricing function f at a time t on an edge $e = uv \in E$ denoted by $f(A_e(t), \lambda(L(u), L(v)), \tau(t))$ is *linear in* A if and only if $f\left(k \times A_e(t), \lambda(L(u), L(v)), \tau(t)\right) = k \times f(A_e(t), \lambda(L(u), L(v)), \tau(t))$.

Definition of "Utility:" A cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *utility* if and only if $\forall e \in E, \forall t \in T$, $q = \delta(t) \rightarrow P_q(e)$ is linear in $A_q(e)$ and $A_q(e) \neq 0 \rightarrow P_q(e) \neq 0$.

In other words, there ain't no such thing as a free allocation, and price is proportional to quantity. The final condition above, i.e., $A_q(e) \neq 0 \rightarrow P_q(e) \neq 0$ ensures that $f(\,)$ is not just the

[5] Frank Harary, *Graph Theory*, Addison-Wesley, 1969.

trivial function $f() = 0$. Note that pricing may still be utility even if it is dynamic: the constraint is that at any given time, price is proportional to allocation.

Definition of "on-Demand:" A cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *on-Demand* if and only if it is completely feasible.

This may seem overly simplistic, but it specifically implies, for any node $v \in V$ and for any $t_1, t_2 \in T$, if $R_v(t_1) \neq R_v(t_2)$ and if $\sum_{u \in I_v} A_{uv}(t_1) + R_v(t_1) - \sum_{x \in O_v} A_{vx}(t_1) \geq 0$ then $\sum_{u \in I_v} A_{uv}(t_2) + R_v(t_2) - \sum_{x \in O_v} A_{vx}(t_2) \geq 0$.

In other words, even if the demand of a node changes instantaneously, the cloud "responds" to meet that demand. (Note that any resource vectors in R or A can have values representing supplies, demands, or mixed.)

Now we can formally define a cloud:

Axiomatic Definition of a "Cloud": A *cloud* is a cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ satisfying 5 axioms:

1. Common,
2. Location-independent,
3. Online,
4. Utility,
5. on-Demand.

Note that this definition of cloud—a cloud structure coupled with these five axioms—provides a precise characterization of the key, intuitive notions in many definitions, such as the NIST definition of cloud[6]:

> [Cloud Structure:]
>
> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
>
> This cloud model promotes availability and is composed of five essential characteristics**,** three service models, and four deployment models."
>
> [Axioms:]
>
> **Essential Characteristics:**

[6] Mell and Grance

> *On-demand self-service.* A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
>
> *Broad network access.* Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).
>
> *Resource pooling.* The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.
>
> *Rapid elasticity.* Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
>
> *Measured Service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability [typically through a pay-per-use business model] at some level of abstraction appropriate to the type of service (e.g.,storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service."

Sometimes, a cloud structure will not meet all the axioms. We can call such a structure a *semicloud*. Often, we will be interested in simple structures consisting of only suppliers and customers, which we can call a *simple* cloud:

Definition: a cloud structure $(\mathbb{S}, \mathbb{T}, G, Q, \delta, q_0)$ is *simple* when $G = (V, E)$ is bipartite, i.e., $V = C + S, C \cap S = \emptyset$, and $e = uv \in E \rightarrow u \in S$ and $v \in C$. Moreover, $s \in S \rightarrow R(s) \geq 0$, and $c \in C \rightarrow R(c) \leq 0$.

In effect, there are customers and suppliers, suppliers have resources and customers need them, and each group only interacts with members of the other. A simple cloud is then an entity that has a simple cloud structure and satisfies the cloud axioms. One other definition that will come in useful in the next section is this:

Definition: An allocation A is *sole-sourced* at time t if and only if

$$\forall v \in V, \forall u, x \in I_v, A_{uv}(t) \neq 0 \text{ and } A_{xv}(t) \neq 0 \Rightarrow u = x$$

5. Research Agenda

This definition of the cloud is at the intersection of various modeling approaches. We have described a few, for example Turing and finite state machines. In addition, there are a variety of problems in operations research that are related: integer programming, linear programming, multiperiod resource allocation[7], multiperiod stochastic optimization and multiperiod stochastic programming[8], network flows and min-cut / max-flow algorithms, and the like.

We will outline proofs of some theorems to show the range of results that can be derived from this model. Elsewhere we have provided extensive proofs of a number of them.

Given a cloud, will be interested in solving different sorts of problems. These may fall into a number of general categories. We are interested in either the solution, or a parametric solution, or an assessment of the computational complexity of the solution. Here's a partial list:

- **Characterization and Isomorphism** – it is clear that *apparently* different clouds may be identical. For example, relabeling vertices may result in a nominally different cloud, but it will not have any different behavior. Also, for any edge, reversing the orientation of the edge and negating the allocation for that edge results in no change to feasibility. One can also show equivalence of certain conditions. For example, the existence of a feasible allocation for a (weakly connected) component $G_i = (V_i, E_i)$ is equivalent to the condition that $\sum_{v \in V_i} R_v \geq 0$. The proof can be constructed along similar lines to the Ford-Fulkerson algorithm[9] to find the maximum flow in a network with a source and a sink. Given that $\sum_{v \in V_i} R_v \geq 0$, we start with all allocations set to $0 =< 0,0, \ldots 0 >$. If this is a feasible allocation, we terminate. Otherwise, we observe that if there is a node, say, v_d. with net demand (i.e., it is under-resourced, accounting for its own capacity and net inflows and outflows) in a specific dimension in $\mathbb{R}^r$, there must be a node v_c somewhere with a net capacity in that dimension. Since G_i is weakly connected, there is always a semipath from v_d to v_c, so we augment the current allocations along the path by the minimum of either the needed resources or the maximum amount of excess resources at the node. For each interior node along the path, there is no net change in status. If there is still net demand at v_d, we find another node with net capacity. We can conduct this process independently, dimension by dimension, and the procedure will terminate successfully within $|V_i| \times r$ steps, each of which has a polynomial time bound.

- **Optimization Problems** – a broad range of problems in operations research in general and linear programming in particular deals with structures such as these. The *transportation* and *transshipment* problems[10] relate to finding a cost-optimal allocation,

[7] Hanan Luss, "Optimization of a Multiperiod Resource Allocation Model," Operational Research Quarterly, Vol. 25, No. 1, March, 1975.
[8] Norio Hibiki, "Multi-period Stochastic Optimization Models for Dynamic Asset Allocation," April 30, 2003, at http://www.ae.keio.ac.jp/lab/soc/hibiki/profile_2/INFORMS_2003_ModelComparison.pdf .
[9] Shimon Even, *Graph Algorithms*, Computer Science Press, 1979.
[10] Frederick Hillier and Gerald Lieberman, *Introduction to Operations Research,* 3rd Edition, Holden-Day, 1980.

given multiple customers and multiple suppliers and a linear cost for each pairwise route between customers and suppliers.

- **Satisfiability and Assignment** – is there an *assignment,* that is, a feasible allocation A, of customers ($R(v) < 0$) to service nodes ($R(v) > 0$) such that the demand from each customer may be met by the service nodes? Usually an assignment is intended to be a matching, that is, each customer is matched exactly with one and only one supplier, however, one can consider variations where each customer is assigned to a single supplier (i.e., sole-sourced), but any supplier may service zero or more customers. I've shown elsewhere[11] that under these conditions, even if the cloud is simple, as long as all allocations are sole-sourced, and the quantity varies by customer, the problem is NP-complete[12]. If quantity doesn't vary by customer, this is polynomial-time transformable into the *maximum cardinality bipartite matching*, or *marriage* problem, for which a P-time algorithm has been devised by Hopcroft and Karp[13], and if more than one service node may satisfy a given customers demand, the problem reduces to the simple decision problem of whether the aggregate quantity of resources is greater than 0, since G is connected and the direction of an arc doesn't restrict the net allocation. There are also optimization forms of the problem where we look for a minimum cost solution.

- **Cloudonomics and Optimization** – we may also ask economic optimization questions, such as finding an optimal allocation given the pricing function P. Elsewhere, I've shown[14] that whenever $D(t)$ fluctuates, a utility solution (P is linear in allocations) or hybrid solution of utility and flat-rate can be economically optimal. I've also shown[15] that there is value to demand aggregation. If n customers have demand $D_1(t), D_2(t), \ldots D_n(t)$, and each $D_i(t)$ has mean $\mu > 0$ and variance σ^2, then each has a coefficient of variation of $\frac{\sigma}{\mu}$, but the aggregate $\sum_{i=1}^{n} D_i(t)$ has mean $n \times \mu$ and variance $n \times \sigma^2$, and therefore has a coefficient of variation of only $\frac{\sqrt{n}\sigma}{\mu}$. In other terms, a penalty function due to overcapacity or undercapacity can be reduced to be $\frac{1}{\sqrt{n}}$ of its unaggregated value.

- **Scheduling and Sequencing** – There are a variety of scheduling problems that are known to be intractable, for example, job sequencing on one processor with release times and deadlines, sequencing to minimize tardy tasks, multi-processor scheduling,

[11] Joe Weinman, "Cloud Computing is NP-Complete," Feb. 21, 2011, http://www.joeweinman.com/Resources/Joe_Weinman_Cloud_Computing_Is_NP-Complete.pdf.
[12] Michael R. Garey and David S. Johnson, *Computers and Intractability, W.H.Freeman & Co., 1979.*
[13] J. Hopcroft and R. Karp, "An $n^{\frac{5}{2}}$ Algorithm for Maximum Matching in Bipartite Graphs," SIAM J. Computing, 1975, pp. 225-231.
[14] Joe Weinman, "Mathematical Proof of the Inevitability of Cloud Computing," January 8, 2011, http://www.joeweinman.com/Resources/Joe_Weinman_Inevitability_Of_Cloud.pdf.
[15] Joe Weinman, "Smooth Operator: The Value of Demand Aggregation," February 27, 2011, http://www.joeweinman.com/Resources/Joe_Weinman_Smooth_Operator_Demand_Aggregation.pdf.

resource constrained scheduling, job-shop scheduling, and so forth.[16] The transition function can encode various constraints and parallelization strategies, and thus create a new class of problems on the cloud.

- **Coverage** – in graph theory, the vertex cover problem, which is one of the core NP-complete problems, may be stated as "is there a set of vertices $V' \leq V, |V'| \leq k$, such that $e = uv \in E$ implies either $u \in V'$ or $v \in V'$?" In addition, we may be interested in what might be called the "cloud cover" problem, where we are looking to ensure that all allocations at all times keep $\lambda \leq j$,so that all customers are within a given distance. Megiddo and Supowit have shown that this general problem is intractable as well. [17]

- **System Dynamics** – Given that customers are active, intelligent agents, having a choice of service nodes with different price plans, and a random initial assignment of customers to service nodes with different plans, how will the system evolve. Elsewhere[18], I've shown that starting from a random allocation A_0 of customers to flat-rate and utility priced service providers in state q_0, rational light users on flat-rate plans will defect (be re-allocated via the transition function δ) to pay-per-use plans, and rational heavy users will defect to flat-rate plans. Over time, the price of the flat-rate plans will increase, driving continued imbalances in favor of the pay-per-use plans. A terminal state is reached where only the heaviest users remain on the flat-rate plans.

- **Performance** – given performance, throughput, latency, or response time objectives, and given a distribution of customers, how can different choices of service node architectures and resourcing impact these objectives. Elsewhere[19] I've shown how dispersion can be traded off against consolidation. Briefly, more service nodes reduces the expected or worst case value between a customer node and a service node, but greater consolidation improves the ability to aggregate and thus smooth demand variation.

- **Fairness, Liveness, Reachability** – given an allocation over time, particularly one that is partly or continuously insufficient, how do we ensure that each customer gets a fair share of the resources that *are* available. How do we evaluate whether nondeterministic transition, location, and allocation rules may lead to states with no possible allocation, whereas other trajectories may lead to completely feasible allocations.

[16] Garey and Johnson

[17] N. Megiddo and K. Supowit, "On the Complexity of Some Common Geometric Location Problems," SIAM J. Computing, Vol. 13, No. 1, February 1984.

[18] Joe Weinman, "The Market for Melons: Quantity Uncertainty and the Market Mechanism," September 6, 2010, http://www.joeweinman.com/Resources/Joe_Weinman_The_Market_For_Melons.pdf.

[19] Joe Weinman, "As Time Goes By: The Law of Cloud Response Time," April 12, 2011, http://www.joeweinman.com/Resources/Joe_Weinman_As_Time_Goes_By.pdf.

6. Extensions, Restrictions, and Modifications

There are numerous definitions of Petri nets that are equivalent to each other. There are also numerous extensions, restrictions and modifications that have modified the original definition, often to meet specific cases: Time Petri nets, inhibitor arcs, zero-testing, constraints, etc.[20]

Similarly, for Turing machines[21], there are numerous variations: single tape, multitape, pushdown stack, etc., that are all equivalent, and, per the Church Hypothesis, equivalent to other computing models and our intuitive notion of a computation.

No doubt a reader of this paper will agree with some elements of the proposed model, grudgingly allow some, vehemently disagree with some, and question obvious omissions. Petri net models have been through dozens, if not hundreds, of such modifications. Here are some natural extensions.

Transformation

One logical extension is the notion of a transformation. Consider modeling a bakery, for example. It has inputs of flour, eggs, butter, sugar, water, and energy, but may have outputs of bagels, brioches, and butterfingers. In a cloud computing context, one may be "buying" an application, but the input resources are servers, storage, and so forth. Two variations suggest themselves. In a first, an $r \times r$ transformation matrix $\mathbf{M}_\mathrm{v}$ applies to a node and its inputs, and we now define feasibility as:

$$\forall v \in V, \mathbf{M}_\mathrm{v}\left(\sum_{u \in I_v} A_u(t) + R_v(t)\right) - \sum_{x \in O_v} A_u(t) \geq 0$$

One can set $\mathbf{M}_\mathrm{v}$ equal to the identity matrix for those nodes without transformations. However, such a node-based approach may fail in modeling some types of systems, because applying the transformation only to the "input" edges implies that there is a unique classification of such an edge as an input edge. However, we would like to preserve the equivalence that any input edge uv with allocation A_{uv} is equivalent to an output edge vu with allocation $A_{vu} = -A_{uv}$. The node based approach fails because an allocative flow that would have had the transformation $\mathbf{M}_\mathrm{v}$ applied no longer does, as it is now classified as an output edge rather than an input edge, destroying the equivalence relationship: we can't guarantee that $\mathbf{M}_\mathrm{u} = {\mathbf{M}_\mathrm{v}}^{-1}$.

Consequently, a better approach is to apply transformations on the edges, rather than at the nodes, even if many or most or all edges have an identical transformation. Flour, yeast, and water, by this convention, are not transformed into bread at the bakery, but, in effect, on the trip

[20] James L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.
[21] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

from the bakery to the oven. M is then a mapping from the set of edges to the set of $r \times r$ matrices on the reals, and for $\mathbf{M}_{\mathrm{uv}} \in M$ when $uv \in E$ and $A_{uv} \in A$ we can equivalently represent things as $\mathbf{M}_{\mathrm{vu}} = \mathbf{M}_{\mathrm{uv}}{}^{-1} \in M$, $vu \epsilon E$ and $A_{vu} = -A_{uv}$. Then, the correct form for the feasibility relationship, whose semantics connote conservation, is:

$$\forall v \in V, \mathbf{M}_{\mathrm{uv}} \sum_{u \in I_v} A_u(t) + R_v(t) - \mathbf{M}_{\mathrm{xv}}{}^{-1} \sum_{x \in O_v} A_u(t) \geq 0$$

As before, setting $\mathbf{M} = \mathbf{I}$, the identity matrix, preserves the *status quo ante*.

Resource Storage Over Time

We'd occasionally like to store variables. For example, one might define a resource level to be the highest level of demand ever previously reached. Some transition functions, however, namely, those based on recursive functions, only consider the prior state, not the entire set of prior states. There are other tricks that one can use, though. For example, it has been shown that the finite state control of a Turing machine can store a finite amount of information. For example, suppose we want to store a bit b, which can be 0 or 1. One way to do this is to replicate the set of states Q, into, in effect, $Q' = Q_{b=0} + Q_{b=1}$.

However, it should be evident that we can create one or more dummy nodes that can store an infinite amount of information. For example, one node v can store a variable x by letting δ set $R_v(t_i) = x$ to "write" the variable x, and then ensuring that δ sets $R_v(t_{i+1}) = R_v(t_i)$, to store the value. It should be apparent that by various encoding strategies, an infinite number of variables may be maintained, even if there is only a single dummy node and r is 1. Moreover, we can "neutralize" the overall effect of this dummy variable in global allocation questions, but pairing it with another dummy variable that is its inverse: $R_{v'}(t_i) = -R_v(t_i)$ together with an allocation between them, $A_{vv'}(t_i) = R_v(t_i)$ with zero cost.

Virtualization

While virtualization is no doubt an important technology, from a formal modeling perspective we can treat its existence either as an additional resource and/or consider that it is implicitly contained in the current proposed cloud structure via the notion that resource quantities may be members of the reals, not just integers: instead of only allocating 1 server or 2 servers, we can allocate 1.37 servers.

Aggregation

We have not specifically distinguished between customers and end-users, or between service providers and their data centers or "supply centers." We can disjointly partition G, or V, into clusters and individual nodes may be clustered into "service providers" and "customers," where each service provider has a set of service nodes, and each customer has a set of users. Of

course, we can partition V however we want, e.g., into federations of suppliers with availability zones containing service nodes, and buying cooperatives comprising firms with divisions or locations comprising users.

This approach can be used to model what NIST calls "deployment models," e.g., private clouds, public clouds, community clouds, and hybrid clouds. In fact, the formal model of a cloud described herein may be useful in generating precise definitions of these deployment models, although one may argue that they are the domain of legal and contractual concerns and the theory of the firm.

7. Business Implications

We note that there are numerous business implications that can be explored from this type of model, which I've shown elsewhere. Here are a few examples.

Under assumptions of two providers, one of which is flat-rate based on the average of its customers demands, and the other is pay-per-use in accordance with the utility axiom, and customers with dispersed levels of demand, a transition function δ will lead to system dynamics effects wherein "virtually all" customers defect to the utility pricing scheme.[22] This has implications for services pricing for rational customers. Moreover, the total spend will not shift. This has implications for revenue impact projections when providers shift pricing plans.

If there is one flat-rate provider and one pay-per-use provider, a customer with variable demand can often optimize spend either through a pure pay-per-use strategy (the entire allocation flows from the pay-per-use node), or through a hybrid allocation.[23]

A penalty function associated with a feasible but not perfect allocation (excess resources) or an insufficient allocation (insufficient resources) can be reduced to a level of $\frac{1}{\sqrt{n}}$ of its value for an individual workload by aggregating workloads when the individual demand functions $D_i(t)$ are independent[24]. This means that midsize providers can reasonably compete with large or infinitely large providers, since n does not need to be very large for the penalty function to get close to zero.

Suppose the transition function δ does not satisfy the on-demand axiom, but still attempts to respond to shifts in $R(t)$ with a new allocation $A(t+\varepsilon)$ after some delay ε in time. The penalty function corresponds to a current imperfect allocation, either feasible or insufficient, and is based on δ and ε. The faster that $D_i(t)$ wanders, and the larger the delay ε, the larger the

[22] Joe Weinman, "The Market for Melons: Quantity Uncertainty and the Market Mechanism""
[23] Joe Weinman, "Mathematical Proof of the Inevitability of Cloud Computing."
[24] Joe Weinman, "Smooth Operator: The Value of Demand Aggregation."

penalty will be. We can characterize the size of the penalty based on $D_i(t)$: which may be linear, a random walk, an exponential (not exponential distribution, but ke^t), etc.[25]

8. Relationship to Other Models

This structure for cloud is motivated by leveraging insights gained in the half century or more of formal model development in areas such as Petri nets and finite state automata, but not limited by it.

As in Petri nets, a graph model of a cloud is important, indicating various nodes who may act as customers, suppliers, or traders who exchange one resource for another. However, Petri nets often distinguish between "places" and "transitions" and also use multi-arcs. We do not need that complexity; although we may later—as an overlay to the core model—choose to partition nodes into various subsets.

Network flow models define capacities along edges, but nodes are unlabeled except for the source node and the sink node. A maximum flow along an edge may be less than or equal to the *defined* capacity, but no more. Here, we place "capacities" on the nodes, rather than the edges. However, flows on edges are still constrained by capacity: namely, the sum of the outflows from a node cannot exceed the sum of the inflows together with the (possibly negative in one or more dimensions) capacity of the node for it to be a feasible flow.

As with formal Turing machine, finite state automata, and Petri net definitions, we are interested in behavior over time. Consequently, we also have a δ function or transition relation. In the same way that a δ function often integrates multiple inputs, or criteria, for example, current head position on the tape, current state, currently read symbol, and then may generate one or more outputs, e.g., head move left or right, a next state, and/or an output symbol, our cloud structure integrates multiple inputs (resources, allocation, location, and pricing) to generate multiple outputs in the same categories.

A Turing or finite-state machine begins its δ function operation in an initial state q_0 and then may move to other "states" within Q. This is, however, a convenient fiction: the true "state" of the machine depends on not only the state of the finite control, but also the state of the tape (what is its current complete, possibly infinite string of written symbols from Σ^*) and the state of the head relative to the tape. This is referred to as an "Instantaneous Description," but today we would recognize it as "state." In our model, the state is the combination of node resource quantities and the current allocation and the location of the nodes and the pricings.

As with network flow models, we use weighted edges. However, the semantics in network flow models involve edge *capacities*, here we are interested in edge *distances*. This is to model the

[25] Joe Weinman, "Time is Money: The Value of 'On-Demand'"

physical dispersion of the cloud, and the possible movement of various nodes which may act as suppliers, customers, or traders.

In Petri nets, the markings evolve in accordance with the transitions, and in Turing machines, the "markings," i.e., symbols on the tape, and the state of the automaton evolve as the machine operates. In our model, there are a set of interacting processes: shifts in supply and demand may cause changes in pricing and in allocation flows, which in turn can influence supply and demand and pricing, and so on.

The definitions of Finite State Automata, Turing machines, and Petri nets, although widely accepted, appear to be somewhat deficient in the completeness of formal modeling in several respects. For example, the basic formal model of a one-way infinite tape does not include either the tape or the notion of time. Another way to look at this is that if we were to write a program that modeled a Turing machine, it would certainly require an array along the lines of "`Tape[k] of Type Symbol`", or else it would not be very useful. One might argue that a counter variable "`Time`" would also be helpful, although not required. Discrete time is, in effect, a result of iteration, and one can argue that discrete steps be explicity recognized (e.g., "`for(time=0;time++;True){ doiteration();}`") or implicitly represent the number of iterations since the initial state, say q_0 (e.g., "`while(True){doiteration();})`. Because we take time for granted, we arguably err either way by including it or excluding it explicitly.

The model of a cloud proposed here encompasses a broad variety of other models: networks, network flow models, Markov processes, Turing machines and other finite state automata, Petri nets, etc. We've highlighted some similarities and differences. What would also be of interest is highlighting equivalences. As an example, since δ can be any computable process, it certainly already is equivalent to a Turing machine. However, it might be of interest to model the tape of a one-tape Turing machine as an infinitely long path of vertices V in an infinite graph G within a cloud structure. A one-dimensional resource vector could encode symbols from the alphabet,, thus $G \cup R$ can model the tape. And the δ function can easily encode the finite state machine logic, "writing" symbols by adjusting resource vectors and moving the head, and ensuring constraints, such as that any tape square, i.e., vertex, has only one valid value at a given time.

9. Summary

The main purpose of this document is to illustrate that appropriate formalisms can be used to rigorously model "clouds," whether in computing or in other domains. Any such model, of course, does not represent the entire complexity of the real world. However, models can help elucidate characteristics that are of use in understanding real world behaviors. It is my hope that if nothing else, this paper is an existence proof that one can develop cloud models that have a meaningful degree of rigor, and that others will be encouraged to extend or modify the one proposed here, or to develop their own.

10. Acknowledgement

I had the good fortune to participate with Prof. Michael Lightner of UC-Boulder at the IEEE Cloud Computing Symposium in Hong Kong, where, in his closing conference remarks he observed that although cloud computing is clearly an active area of research, there was a dearth of formal models. I would like to gratefully acknowledge that since then, he has generously given of his time and formidable insight in improving this model, suggesting extensions and variations, questioning gaps and weaknesses. Of course, any faulty logic, errors or omissions are my own.

Appendix

We are modeling time-varying systems and processes. By system and process, we can use the following eloquent definitions from Bart Meijer:

> A "process" is a sequence of transformations that cause the input(s) of the process to be converted into a desirable output. A "system" is a meaningful set of elements and relations. The relations represent both interactions between elements and interactions with the environment of the system.[26]

We often use terms like sets, functions, and processes informally. However, they also have formal definitions, which we will occasionally refer to.

Briefly, a set is a collection of objects, e.g., $\{a, b, c\}$, where each object appears at most once. An ordered pair, e.g., $< a, b >$, lets us impose order on the objects: $< a, b >$ is different than $< b, a >$, unless $a = b$, but may be considered to be a set, for example, $< a, b >= \{\{a\}, \{a, b\}\}$. A vector in an ordered tuple, which also may be considered as a set using a similar construct. While we may not normally think of it in this way, a function $y = f(x)$ is also just a set of ordered pairs, for example, the function $y = x^2$ on the natural numbers is the set $f = \{< 1,1 >, < 2,4 >, < 3,9 >, \ldots\}$. Finally, a *process* may be viewed as a type of function that takes one or more inputs and generates one or more outputs. Such a process may be thought of as a finite or infinite set of relations among elements that relate input(s) to output(s), or as a computation that generates such relationships dynamically. For example, the "process" of interchanging two natural numbers can be viewed as a relation between $\mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$, which in turn is just a countable set of 4-tuples $\{< input_1, input_2, output_1, output_2 >\}$, e.g., $\{< 1,1,1,1 >, < 1,2,2,1 >, < 2,1,1,2 >, < 1,3,3,1 >, < 3,1,1,3 >, < 2,3,3,2 >, \ldots\}$, or a function from $\mathbb{N}^2$ onto itself.

We want to model resources and resource allocations. In simple models, we may only have one type of resource. In others, we may have many. If there are r different types of resources, we will use values from the vector space $(\mathbb{R}^r, \mathbb{R})$, which is the real-valued r-dimensional vector space on the field of reals. As is customary, we use 0 as shorthand for $< \underbrace{0,0, \ldots 0}_{r\ times} >$.

In a vector space, the relation "≤" is not straightforward. When there is a norm on the vector (e.g., "distance" on the plane, where $||V|| = \sqrt{{v_1}^2 + {v_2}^2}$), there is a mapping from vectors to reals, and reals have the relation "≤" defined. However, this is not useful to us, as the different dimensions in our space signify different objects, e.g., servers and storage. For our purposes, we will use a definition of "≤" that makes Y a partially ordered vector space, namely that $< u_1, u_2, \ldots u_r > \leq < v_1, v_2, \ldots v_r >$ if and only if $u_1 \leq v_1,\ u_2 \leq v_2, \ldots\ u_r \leq v_r$. This satisfies the

[26] Bart R. Meijer, "Reducing Complexity through Organisational Structuring in Manufacturing and Engineering." Manufacturing Complexity Network Conference April 2002, University of Cambridge.

requirements for a partially ordered vector space, namely that $u \leq v \rightarrow u + a \leq v + a$, and that if $0 \leq a, u \leq v \rightarrow u \times a \leq v \times a$.

Formal models of computation are often defined using n-tuples. For example, a deterministic finite state automaton can be defined[27] as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, Σ is the input alphabet, q_0 is the initial state, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of final states.

A Turing machine, which is like a finite state automaton but not only "reads" symbols and thus "recognizes" strings, but can also "writes" symbols and thus can calculate ("recursively enumerate.") Thus, a Turing machine adds two more components: Γ and B. Γ, the set of output symbols, which is a superset of Σ and also includes the blank symbol B.[28]

A marked Petri net can be defined[29] as a 5-tuple (P, T, I, O, μ), where P is a set of places, T is a set of transitions, I is an input function, O is an output function, and $\mu: P \rightarrow \mathbb{N}_{\mathbf{0}}$, i.e., maps places to the non-negative integers. Surprisingly, most formal models of Petri nets seem to ignore formalizing a transition function that is at the heart of "executing" or "firing" a Petri net, and most models in general shy away from formalizing time. Although perhaps overly rigorous, we do both here.

A group[30] is a mathematical structure comprising a set and an operator, satisfying the group axioms: closure, associativity, identity, and invertibility, and if it is Abelian, commutativity. Similarly, we have defined a cloud as a mathematical / graph theoretic structure that satisfies the cloud axioms: common, location-independent, online, utility, and on-demand.

[27] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
[28] Hopcroft and Ullman.
[29] James L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.
[30] I. N. Herstein, Topics in Algebra, 2nd Ed., John Wiley & Sons, 1975.

