Cloud Computing is NP-Complete

Working Paper, February 21, 2011

Joe Weinman[1]

Permalink: http://www.JoeWeinman.com/Resources/Joe_Weinman_Cloud_Computing_Is_NP-Complete.pdf

Abstract

Cloud computing is a rapidly emerging paradigm for computing, whereby servers, storage, content, applications or other services are provided to customers over a network, typically on an on-demand, pay-per-use basis. Cloud computing can complement, or in some cases replace, traditional approaches, e.g., owned resources such as servers in enterprise data centers.

Such an approach may be considered as the computing equivalent of renting a hotel room rather than owning a house, or using a taxi or rental car rather than owning a vehicle. One requirement in such an approach is to determine which resources should be allocated to which customers as their demand varies, especially since customers may be geographically dispersed, cloud computing resources may be dispersed, and since distance *may* matter due to application response time constraints, which are impacted by network latency.

In the field of computational complexity, one measure of the difficulty of a problem is whether it is *NP-complete (Non-deterministic Polynomial-time complete)*. Briefly, such a designation signifies that: 1) a guess at a solution may be verified in polynomial time; 2) the time to solve the problem is believed to grow so rapidly as the problem size grows as to make exact answers impossible to determine in meaningful time using today's computing approaches; 3) the problem is one of a set of such problems that are roughly equivalent to each other in that any member of the set may be transformed into any other member of the set in polynomial time, and solving the transformed problem would mean solving the untransformed one.

We show that an abstract formulation of resource assignment in a distributed cloud computing environment, which we term the CLOUD COMPUTING demand satisfiability problem, is NP-complete, using transformations from the PARTITION problem and 3-SATISFIABILITY, two of the "core" NP-complete problems. Specifically, let there be a set of customers, each with a given level of demand for resources, and a set of servers, each with a given level of capacity, where each customer may be served by two or more of the servers. The general problem of determining whether there is an assignment of customers to servers such that each customer's demand may be satisfied by available resources is NP-complete.

The impact on Cloudonomics is that even if resources are available to meet demand, correctly matching a set of demands with a set of resources may be too complex to solve in useful time.

---

[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

There are a number of definitions of cloud computing. At a high level, however, we can imagine a group of customers interacting with a set of servers on a globally ubiquitous basis over a network. We use the term "server" as simplified shorthand for "data center," "service center," "service node," "resource cluster," and the like. In some cases these customers may be individuals, using services such as search, web mail, or social networking services, or they may be viewed as enterprises, procuring services such as computing resources on a pay-per-use, on-demand basis.

FIGURE 1: A Global Cloud Computing Environment

To deliver services and resources on demand over a network requires solving numerous technological problems, including automated provisioning, dynamic virtual server migration, network security, and so forth. However, the problem we focus on in this paper is a deceptively simple one: matching demand for resources arising from customers with resource capacity available at the servers. We can create a simple model that looks like this, showing that each customer has some level of demand d_i, and each server has some level of resources r_i:

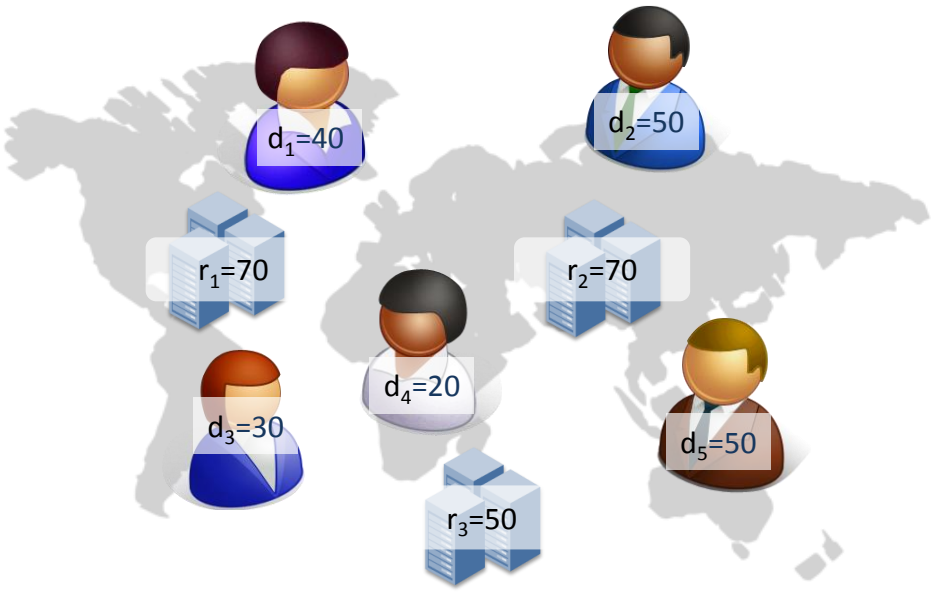

FIGURE 2: Cloud Computing with Quantified Demand and Resources

Also, we may want to address the fact that not all servers may be available to all customers, due to reasons such as network latency, commercial agreements, or security. We can indicate that a customer may be served by a server using a directed edge, as shown below.

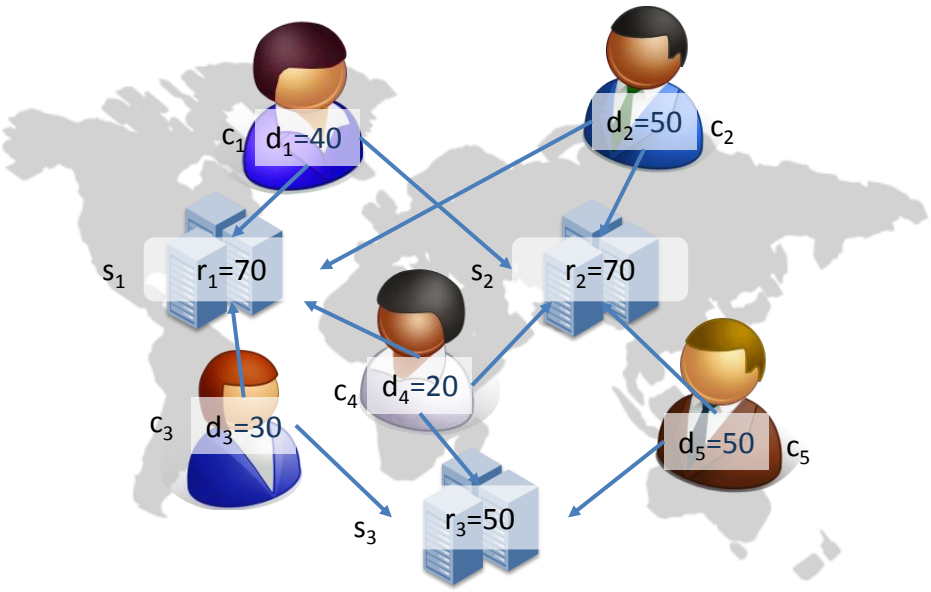

FIGURE 3: Demand, Resources, and Connectivity

Finally, we may abstract the problem as a bipartite graph of servers s_i and customers c_i, with resource requirements or demand d_i and resource availability r_i, as shown:

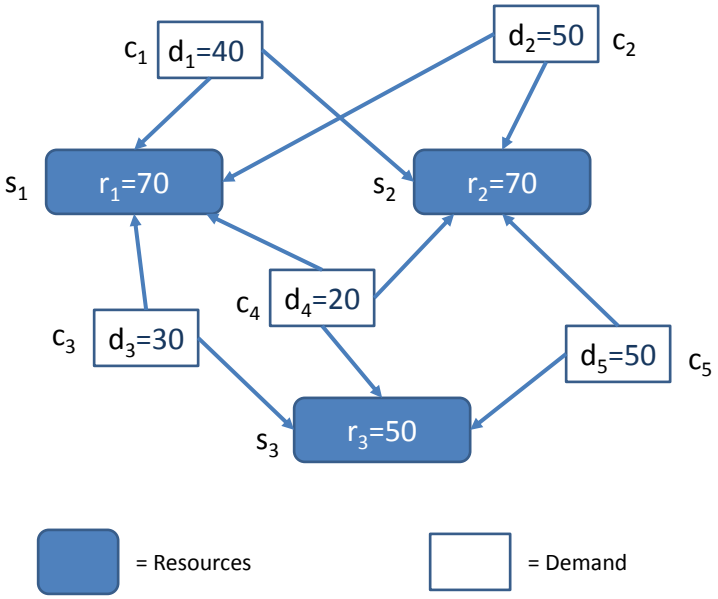

FIGURE 4: A Cloud Computing Graph with Customers Connected To Servers

We can formalize these notions as follows:

Customers and Resource Demand: Let C be a finite set of m customers, i.e., $C = \{c_1, c_2, \ldots c_m\}$. Let $D=\{d_1, d_2, \ldots d_m\}$ be a set of m finite positive demands, where each customer c_i has demand d_i. Elsewhere[2], I have treated each customer's demand as a function of time, e.g., $d_1(t), d_2(t)$, etc., but here we will just treat each demand d_i as a constant, which we can interpret as non-varying demand or an instantaneous snapshot of variable demand.

Generally, we can view each customer's demand as being multi-dimensional, for example, customer c_1 may need 5 processing cores, 2 gigabytes of memory, and 50 gigabytes of storage, but for the purposes of this paper we will treat the demand as one-dimensional.

Servers and Resource Capacity: Demand is served by n distributed servers $S = \{s_1, s_2, \ldots s_n\}$. Let $R = \{r_1, r_2, \ldots r_n\}$ be n finite positive resources, where each server s_i has a quantity of resources r_i. As with demand, generally speaking, we could consider the quantity of resources as varying over time, e.g., $r_1(t), r_2(t)$, etc., and being multi-dimensional. However, here we will treat the resources as constant and one-dimensional.

In this paper, we will assume that demand from any customer must either be served in its entirety by a specific server, or not at all. This is a reasonable assumption, as applications often require a closely coupled set of components that must be physically co-located due to performance reasons.

We would like to understand whether there are sufficient resources to service the demand. If all the customers and servers[3] are in a single location, e.g., a single data center, then this problem is easy to solve. We just need to know whether $\sum_{i=1}^{m} d_i \leq \sum_{i=1}^{m} r_i$, that is, whether the total quantity of demand is less than or equal to the total quantity of resources

And, if latency were not an issue and bandwidth were free, then even if customers and servers were in widely scattered locations, we would be fine with the same constraint.

However, in the case of cloud computing, we can assume that location does matter. To a customer, a cloud computing service or resources appear to be ubiquitous and thus location-independent: after all, a service that could only be accessed from, say, 1234 Main Street in Topeka would not typically be considered "cloud computing." However, this is because behind the scenes there is sufficient available capacity "nearby" enough to meet latency constraints at a manageable level of cost. And, there may be additional constraints whereby not all servers will be suitable for all customers, such as country privacy compliance, secure subnetworks, or other constraints. This implies that servers and customers are less than fully connected.

Consequently, we consider C and S not just as sets, but as members of a bipartite acyclic directed graph $G = \{C \cup S, E\}$, $E: C \rightarrow S$, such that $1 \leq i \leq m \rightarrow \exists j \mid (c_i, s_j) \in E$. In other words,

[2] Joe Weinman, "Time is Money: The Value of 'On-Demand'," http://www.joeweinman.com/Resources/Joe_Weinman_Time_Is_Money.pdf

[3] We use the terms "customers" and "servers" because "client-server" has other connotations.

every customer has one or more, but not necessarily all, servers with resources that can potentially service its demand.

If for each customer there is exactly only one potential server, i.e., the conditions are as described above and $|E| = |C|$, then the problem is again easy to solve, since customers and servers partition themselves into disjoint subgraphs $S_1, S_2, ..., S_k$ and we merely need to meet the condition that for each such subset S, $\sum_{i,c_i \in S} d_i \leq r_j$, given that $s_j \in S$.

Such a circumstance historically occurred in a traditional enterprise glass house data center environment where each employee's terminal could only access services or resources within a nearby corporate data center, rather than say, a remote data center, a public cloud or a *different* enterprise's data center. In such a case, customers and servers partition themselves into multiple "star" clusters, with a server at the center of the star and customers at the points of the star.

Also, if there is more than sufficient capacity at each center to handle workloads, then *any* assignment of customers to resources is likely to work. However, such ubiquitous excess capacity is economically inefficient.

The problem becomes interesting, and surprisingly complex, when each customer can access resources at two or more servers, and there is sufficient, but not excessive, aggregate capacity. As we will show, this problem is NP-complete, that is, unlikely to be solvable in an amount of time that is a polynomial function of m and n. However, a potential solution generated at random may be verified in polynomial time, and this problem may be "transformed" or "reduced" into a set of other computationally complex equivalent problems that have been extensively studied, in the same way that the problem of "getting to work" may be reduced to the problem of "remembering where you left the car keys."

The classic text in the field of computational complexity is *Computers and Intractability,* authored by Michael R. Garey and David S. Johnson[4], then members of the technical staff at the Mathematics Research Center of Bell Labs. We will refer to this book throughout this paper.

The canonical NP-complete problem is SATISFIABILITY, wherein it is desired to be known whether there exists at least one *assignment* of $True$ or $False$ to each variable in a set of Boolean variables such that a Boolean expression—which may be structured as the conjunction ("and"-ing) of a set of disjunctive ("or"-ing) clauses—can be *satisfied,* i.e., can be made to equate to $True$. For example, the expression "(A or B) and (not B or C)" can be satisfied in several ways, including by setting A to $True$ and B to $False$. While this was easy to solve, the general problem, when each clause has three or more Boolean variables, is NP-complete.

In a similar vein, the CLOUD COMPUTING[5] demand satisfiability problem may be expressed as the problem of whether there exists at least one *assignment* of customers to servers such that

[4] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.

[5] We use "CLOUD COMPUTING" to refer to the abstract problem, and "Cloud Computing" to refer to the industry.

the aggregate demand from each customer assigned to a server can be *satisfied*, i.e., can be resourced, by the capacity at that server.

Let customers $C = \{c_1, c_2, \ldots\ c_m\}$ have demand $D=\{d_1, d_2, \ldots\ d_m\}$ respectively, and let servers $S = \{s_1, s_2, \ldots\ s_n\}$ have resources $R = \{r_1, r_2, \ldots\ r_n\}$ respectively. Let $G = \{C \cup S, E\}$ be a bipartitite graph where an edge $e = \{c_i, s_j\}$ signifies that some or all of capacity r_j may be used to serve demand d_i.

With this in mind, we can state the CLOUD COMPUTING problem as deciding whether there is an *assignment* $T \subseteq E$, that is, a subset of the set of edges E, where $|T| = m$, such that

1) Given any customer c_i there exists exactly one server s_j such that the edge $\{c_i, s_j\} \in T$.

2) The sum of the demand D_j^+ arising from the customers served by server s_j is less than or equal to its capacity, i.e., $\forall j, 1 \leq j \leq n$ we have $r_j \geq D_j^+ = \sum d_i\ , \forall i, \{c_i, s_j\} \in T$.

In the example instance we have been using, one such assignment is shown below.

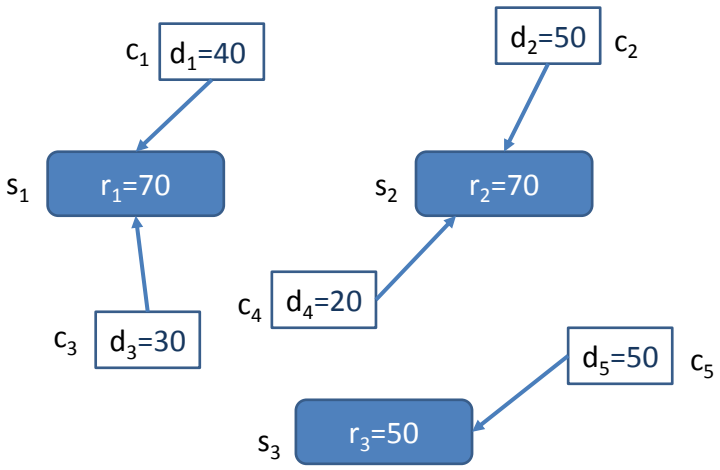

Figure 5: An Assignment of Customers to Servers Satisfying Demand

2. The PARTITION Problem

Garey and Johnson define the PARTITION problem, originally listed in Karp[6], as this: given a finite set A and a "size," or "weight"[7] $w(a)$ associated with each element of A, i.e., $w(a) \in Z^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$?

We can think of A' and $A - A'$ as a partition of A into two disjoint sets A_1 and A_2, where $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$. For example, if the weights of the elements of A are of sizes $1, 2, 3, 4, 5, 7$ and 8, then A may be partitioned into A_1 and A_2 such that the weights of the elements of A_1 are $3, 4$ and 8, and the weights of the elements of A_2 are $1, 2, 5$, and 7 since then $3 + 4 + 8 = 15 = 1 + 2 + 5 + 7$. These kinds of problems show up often in the real world, for example, how can we choose players for two teams so that they have "equal" talent and thus make the game interesting, or how can we divide furniture among two moving vans so that neither exceeds highway weight restrictions.

Note that while the "elements" of A are unique, and thus A is a set, the weights don't need to be. However, if there are two elements with the same size, say, 5, we can assign one of the elements to the first set and the other to the second one and we have simplified the problem.

Garey and Johnson refer to PARTITION as one of the 6 "basic core" NP-complete problems, together with Boolean 3-SATISFIABILITY, 3-DIMENSIONAL MATCHING, VERTEX COVER, CLIQUE, and HAMILTONIAN CIRCUIT. We will show that an instance of the PARTITION problem can be transformed in polynomial time to an instance of the CLOUD COMPUTING problem.

> ***Proposition 1***: PARTITION is NP-complete.
>
> ***Proof:*** See Garey and Johnson, pp. 60-62, for the proof, which is based on a transformation from 3-DIMENSIONAL MATCHING.■

3. CLOUD COMPUTING Demand Satisfiability is NP-Complete

The proof is a formalization of the insight that the CLOUD COMPUTING problem is equivalent to the PARTITION problem. Using the example of A_1 and A_2 above, we can think of that instance of PARTITION as corresponding to the following CLOUD COMPUTING graph:

[6] Richard M. Karp, "Reducibility Among Combinatorial Problems," 1972, at http://www.cs.berkeley.edu/~luca/cs172/karp.pdf

[7] We slightly change notation to use weight $w(a)$ rather than size $s(a)$ to prevent confusion with servers s_i in the present exposition.

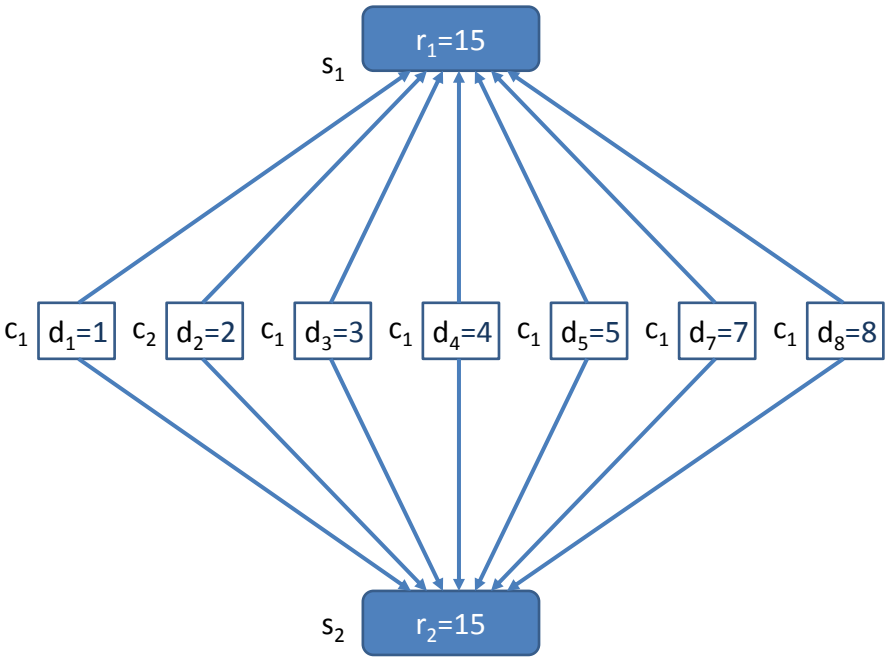

Figure 6: A Transformation of an Instance of PARTITION to CLOUD COMPUTING

And an assignment T which satisfies this instance is

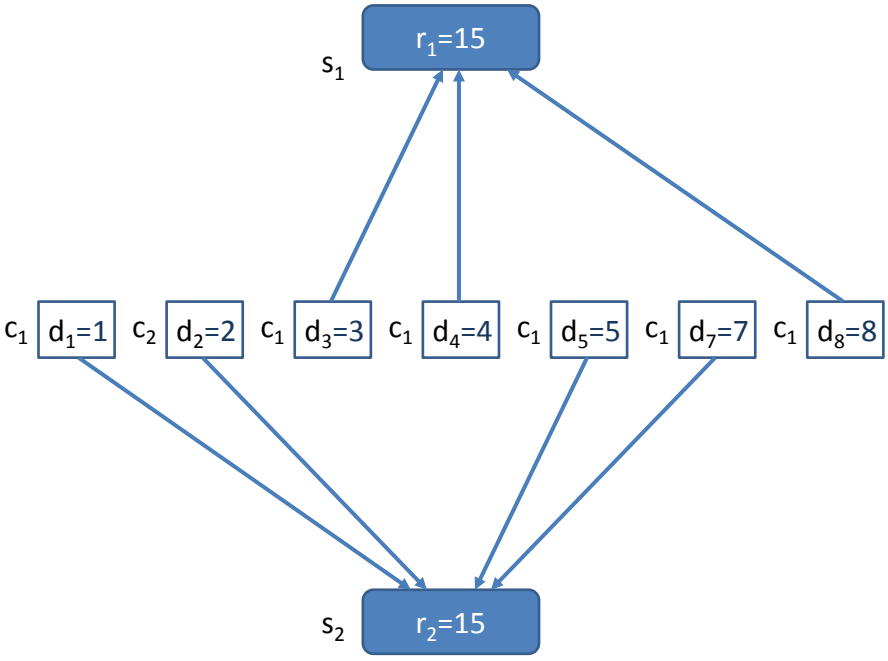

Figure 7: A Solution to an Instance of PARTITION via CLOUD COMPUTING

For a formal proof that CLOUD COMPUTING is NP-complete, we need to show that

1) CLOUD COMPUTING is $\in NP$

2) There is a transformation of any instance of a known NP-complete problem, in this case, PARTITION, into an instance of CLOUD COMPUTING that is satisfiable if and only if there is a solution to the original instance of PARTITION

3) This transformation from PARTITION to CLOUD COMPUTING can be done in polynomial time

We now show this.

Proposition 2: CLOUD COMPUTING is NP-complete.

Proof: Clearly, CLOUD COMPUTING is in NP. If we guess a solution to the problem, we can validate in time that is a polynomial function of m and n that the sum of each of the customer demands served by each server is within its quantity of resources, i.e., that $D_j^+ \leq r_j, 1 \leq j \leq n$.

We now transform PARTITION into CLOUD COMPUTING. Let the instance of PARTITION be a finite set $A = \{a_1, a_2, \ldots a_m\}$ and a weight $w(a_i) \in Z^+$ associated with each $a_i \in A, 1 \leq i \leq m$. We want to know if there is a partition of A into two disjoint subsets $A_1, A_2 \subseteq A$, where $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$, such that $\sum_{a \in A_1} w(a) = \sum_{a \in A_2} w(a)$.

We construct a CLOUD COMPUTING instance $\{C,\ S,\ E,\ D,\ R\}$ as follows. Let C comprise m customers $c_1, c_2, \ldots c_m$, and let D comprise the corresponding demands $d_1, d_2, \ldots d_m$, where each d_i is set to $w(a_i)$. Let S comprise two servers, s_1 and s_2. Let $r_1 = r_2 = \frac{1}{2}\sum_{a \in A} w(a)$. Finally, let $E = \{\{c_1, s_1\}, \{c_2, s_1\}, \ldots \{c_m, s_1\}\} \cup \{\{c_1, s_2\}, \{c_2, s_2\}, \ldots \{c_m, s_2\}\}$.

It is clear that such a transformation can be performed in polynomial time, proportional to m to construct C and D, proportional to a constant to construct S and proportional to m to determine the values of r_1 and thus r_2, and proportional to $2 \times m$ to construct E.

Finally, we note that an instance of CLOUD COMPUTING so constructed has an assignment that satisfies the demand if and only if the corresponding PARTITION problem has a solution. If there is a solution of PARTITION, where $a_i \in A_j$, then we let $\{c_i, s_j\} \in T$ but $\{c_i, s_{3-j}\} \notin T$. Since the sum of the weights $\sum_{a \in A_1} w(a) = \frac{1}{2}\sum_{i=1}^{m} d_i$, we know that there is sufficient capacity r_1 at server s_1 to handle the demand D_1^+, and vice versa, and the same holds for capacity r_2 at server s_2 to handle the demand D_2^+. The converse is also true, since if there is an assignment that satisfies the resource constraints at both s_1 and s_2, and since the demand is indivisible, we have also found a suitable solution to the original PARTITION problem. ■

4. 3-SATISFIABILITY

It will be noted that in the proof above, the topology, i.e., connectivity, of the Cloud Computing graph did not enter into the complexity of the solution in a major way. In a way, this is very powerful, as it shows that there is a fundamentally challenging problem at the heart of CLOUD COMPUTING. However, it is "unsatisfying" (no pun intended) in a way, as real-world considerations of proximity and latency really weren't used that would constrain connections and thereby potentially define a less than fully connected graph. Also, there are what one might term "philosophical" issues with the use of PARTITION, as Garey and Johnson discuss on pp. 90-92 in a discussion regarding "strong" NP-completeness, in that part of the difficulty in solving PARTITION problems is that very large input numbers may be used, and one may need to consider how efficiently those numbers may be encoded in determining the complexity of the problem. In the construction of the prior proof, all m customers were connected to all (i.e., both) servers. In a second proof in the next section, the actual topology of the graph plays a major role in encoding the logic of the underlying complexity. There are "truth-setting" components, "satisfaction-testing" components, each with local behavior, and "communications links" that create a global topology mirroring some of the trade-offs found in the real world. For example, if one runs a large job at a computing center that is nearby, it may then displace another job to a different location, which may then create a domino effect of sorts.

We first refresh the reader's memory regarding the "mother" of all NP-complete problems, 3-SATISFIABILITY:

Let $H = \{h_1\ h_2, \ldots h_p\}$ be a set of Boolean disjunctive clauses on a finite set $U = \{u_1, u_2, \ldots\ u_q\}$ of Boolean variables, such that $|h_i|$=3 for $1 \leq i \leq p$. H is satisfiable if there is an assignment $U \rightarrow \{True, False\}$ such that each h_i is satisfied, i.e., has at least one term that is $True$.

> ***Proposition 3:*** 3-SATISFIABILITY is NP-complete.
>
> ***Proof:*** See Garey and Johnson, pp. 48-50, for a proof via a transformation from SATISFIABILITY.■

5. CLOUD COMPUTING Demand Satisfiability is NP-Complete, Redux

We again show that CLOUD COMPUTING is NP-complete, this time via a transformation from 3-SATISFIABILITY. This proof primarily uses truth-setting components and satisfaction-testing components, and is conceptually related to the proof of the NP-completeness of the VERTEX COVER problem described in Garey and Johnson.

We use the example below to help illustrate the construction.

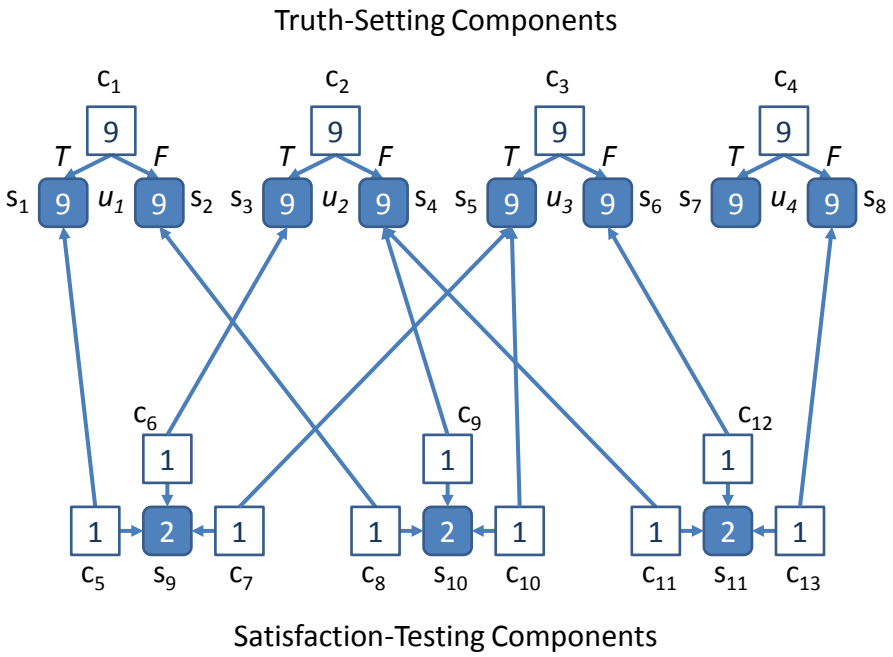

Figure 8: A Transformation of an Instance of 3-SATISFIABILITY to CLOUD COMPUTING

This example CLOUD COMPUTING instance is a transformation from an instance of 3-SATISFIABILITY corresponding to four Boolean variables $U = \{u_1, u_2, u_3, u_4\}$ and $H = \{\{u_1, u_2, u_3\}, \{\overline{u_1}, \overline{u_2}, u_3\}, \{\overline{u_2}, \overline{u_3}, \overline{u_4}\}\}$ or equivalently,

$$True = (u_1 \vee u_2 \vee u_3) \wedge (\overline{u_1} \vee \overline{u_2} \vee u_3) \wedge (\overline{u_2} \vee \overline{u_3} \vee \overline{u_4}).$$

As can be seen, on the top row are "truth-setting" components, e.g. the cluster $\{c_1, s_1, s_2\}$. On the bottom row are "satisfaction-testing" components, e.g., the cluster $\{c_5, c_6, c_7, s_9\}$.

The demand $d_1 = 9$ from customer c_1 can either be met by server s_1 or server s_2. However, once that demand is met, whichever server has served it is now out of resources. The remaining server, however, has enough resources to serve as many customers on the bottom row as it is connected to. For example, if demand from c_1 is met by s_2, then s_2 cannot also serve c_8, however, s_1 will still be able to meet the demand from c_5. Conversely, if demand from c_1 is met by s_1, then s_1 cannot also serve c_5. However, s_2 will have sufficient capacity to meet the demand from c_8.

Turning to the satisfaction-testing components, consider the cluster $\{c_5, c_6, c_7, s_9\}$ on the lower left. Server s_9 has resources $r_9 = 2$. Consequently, it has sufficient capacity to service demands $d_i = 1$ arising from any zero, one, or two—but not all three—of c_5, c_6, and c_7. Therefore, for demands d_5, d_6, and d_7, at least one *must*, but potentially two or three *can* get served from one of the servers in the truth-setting row. The equivalence with 3-

SATISFIABILITY is clear. At least one Boolean term in each clause *must* be served via the truth-setting servers, and as many as three *may* be. However, the truth-setting components are "forced" to make a decision as to a single value—$True$ or $False$—for each Boolean variable.

There may be zero, one or more assignments that satisfy a Boolean expression. In this example, one such assignment is $u_1 = True, u_2 = False, u_3 = True,$ and $u_4 = False$, or, graphically:

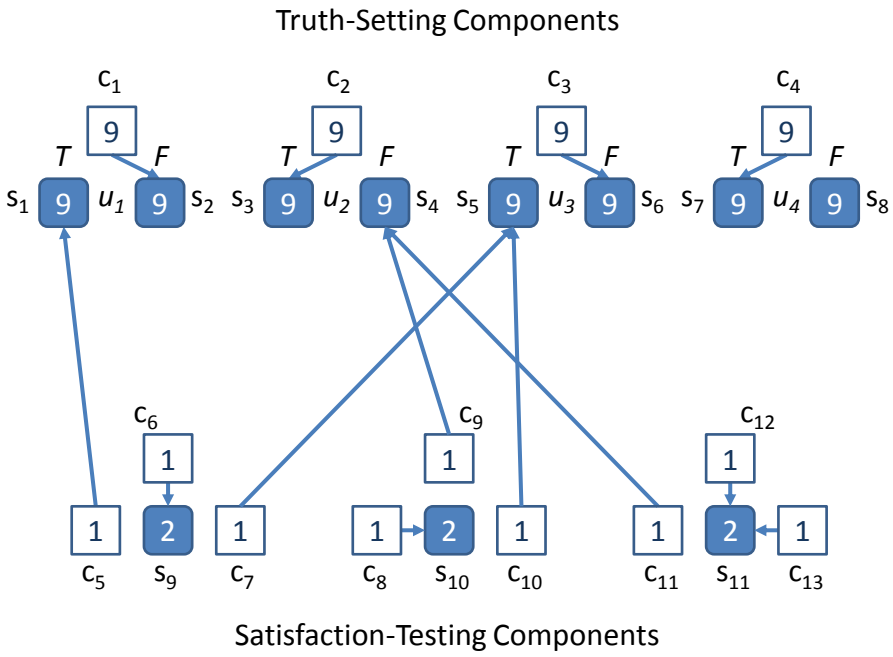

Figure 9: A Solution to an Instance of 3-SATISFIABILITY via CLOUD COMPUTING

Note that demand from switches c_1 to c_4 is resourced by the "opposite" truth value, e.g., in the assignment shown in *Figure 9*, c_1 being resourced by s_2 signifies that u_1 is set to $True$, as any literals in the clauses below may be now easily served by s_1.

With this example in mind, we now turn to a second proof.

> ***Proposition 4:*** CLOUD COMPUTING is NP-complete.
>
> ***Proof***: As in the proof of Proposition 1, CLOUD COMPUTING is clearly in NP. Given a non-deterministic guess as to a solution, we can verify it in polynomial time. We now show a transformation from 3-SATISFIABILITY.
>
> Let $H = \{h_1, h_2, \ldots h_p\}$ be a set of Boolean disjunctive clauses on a finite set $U = \{u_1, u_2, \ldots u_q\}$ of Boolean variables, such that $|h_i| = 3$ for $1 \leq i \leq p$. Construct an instance $\{C,\ S,\ E,\ D,\ R\}$ of CLOUD COMPUTING as follows:
>
> 1) Create q truth-setting switches $c_i, 1 \leq i \leq q$, with demand $d_i = 3 \times p$

2) Create $2q$ truth selections $s_i, 1 \leq i \leq 2 \times q$, each with resources $r_i = 3 \times p$

3) Create an edge from each $c_i, 1 \leq i \leq q$ to s_{2i}

4) Create an edge from each $c_i, 1 \leq i \leq q$ to s_{2i-1}

5) Create $3p$ clause elements $c_i, q+1 \leq i \leq q+3 \times p$, each with demand $d_i = 1$

6) Create p clause gap-fillers $s_i, 2 \times q+1 \leq i \leq 2 \times q+p$, each with resources $r_i = 2$

7) Create an edge from each clause element c_{q+3i+1} to clause gap-filler $s_{i+2q+1}, 0 \leq i \leq p-1$

8) Create an edge from each clause element c_{q+3i+2} to clause gap-filler $s_{i+2q+1}, 0 \leq i \leq p-1$

9) Create an edge from each clause element c_{q+3i+3} to clause gap-filler $s_{i+2q+1}, 0 \leq i \leq p-1$

10) For the 1st, 2nd, and 3rd term in each clause in H, do as follows: if the kth term in clause h_j is u_i, create an edge from $c_{q+3(j-1)+k}$ to s_{2i-1}

11) For the 1st, 2nd, and 3rd term in each clause in H, also do as follows: if the kth term in clause h_j is $\overline{u_i}$, create an edge from $c_{q+3(j-1)+k}$ to s_{2i}

This is a somewhat tedious way of describing the construction illustrated in the example. We note that this transformation may be concluded in time that is a polynomial function of p and q. Finally, we note that per the discussion and the construction there is an assignment to U that satisfies H if and only if there is a $T \subseteq E$ satisfying $\{C, S, E, D, R\}$.

To see this, we note that due to the fact that demand d_i from the switches $c_i, 1 \leq i \leq q$, always is at a level of demand $d_i = 3 \times p$. Since for each switch $c_i, 1 \leq i \leq q$, there is only the ability to have demand served in total either by selections s_{2i} or s_{2i-1}, each *also* with resources $r_i = 3 \times p$, the truth-setting cluster follows the rule that *exactly one and only one* server per cluster will have resources available for use by the satisfaction-testing components, and because $3 \times p$ resources are made available, even if every cluster had every term set to be resourced by that selection, it would have sufficient resources, since there are at most p clauses with at most 3 terms.

As an aside, this is where $3 \times p$ arises in the construction. In fact, if we restrict each clause to at most one occurrence of a term, then we could set demands and resources in the truth-setting components to just p, or even create a custom number of resources equal to the maximum number of occurrences of either a or $\bar{a}$ in the Boolean expression.

It is also clear that since each satisfaction-testing component has three customers each with a demand for 1 resource, but a "local" server only with 2 resources, at least one customer in each satisfaction-testing component must "look elsewhere" beyond the satisfaction-testing component for a resource to be served. Consequently, at least one customer (i.e., variable) in each clause must be "True."

Conversely, from the construction, it is clear that if there is a solution to the instance of 3-SATISFIABILITY, there is a corresponding solution to the instance of CLOUD COMPUTING. Given a truth assignment of $True$ or $False$ to variable u_i, we set each truth-setting component accordingly so that the "switch" customer gets its resources from the selection customer corresponding to $\overline{u_i}$. This leaves sufficient resources in the selection labeled with the same truth value as u_i. Since there is a satisfying assignment to the instance of 3-SATISFIABILITY, this means that each satisfaction-testing component has at least one term that is satisfied, thus can find at least 1 resource from the truth-setting layer. Therefore, there are at most 2 resources needed by the satisfaction-testing component, which can be satisfied by the "gap-filling" server.■

6. Discussion

This paper shows that computationally complex problems can arise under simple architectures such as geographically dispersed resources accessed by geographically dispersed customers.

In the real world, there is usually likely to be sufficient capacity that most or all customers can be served most or all of the time. However, as latency constraints cause demand to need to be fulfilled by "nearby" servers, and user experience or business requirements reduce the ability of those applications to be deferred to whenever resources may become available, this illustrates the challenge of maintaining a cost-effective, minimal set of resources and routing (assigning) customers to those resources for highly interactive workloads. Moreover, in the real world, such a problem must be solved continuously, as servers become available or go off-line, and customer demand levels shift.

Perhaps most interestingly, this brief analysis shows that even under simple assumptions regarding demand and resources, computationally complex challenges can arise. My prior papers in Cloudonomics have looked at issues such as the relative cost of on-demand resources as well as the timeliness of resource provisioning and de-allocation. This paper shows that, at least theoretically, the complexity of matching demand to resources in a distributed, efficient environment can be an issue as well.

