Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.2, January 2016 – www.caeaccess.org

Given a Static Workload Cloud Computing Patterns does it have an Elastic Scaling?

Ravi (Ravinder) Prakash. G
Senior Professor Research,
BMS Institute of Technology
Dodaballapur Road,
Avalahalli, Yelahanka,Bangalore – 560 064

Kiran M
Research Scholar
School of Computing & Information Technology
REVA University, Yelahanka, Bengaluru - 560064

ABSTRACT
Measurability is a concept in elastic scaling based on the following two conditions: (a) a cloud service provider should be cautious, that is, should not exclude any cloud consumer's resource pooling pattern strategy from consideration; and (b) a cloud service provider should consider the cloud consumers' resource pooling pattern preferences, that is, should deem a cloud consumer's resource pooling pattern strategy k_i infinitely more likely than k'_i if it premises the cloud consumer to prefer k_i to k'_i. A resource pooling pattern strategy is measurable if it can optimally be chosen under common resource pooling pattern conjecture in the events (a) and (b). In this paper we present an algorithm that for every finite elastic scaling operation computes the set of all measurable resource pooling pattern strategies. The algorithm is based on the new idea of a Static Workload preference limitation, which is a pair (k_i, V_i) consisting of a resource pooling pattern strategy k_i, and a subset of resource pooling pattern strategies V_i, for cloud service provider i. The interpretation is that cloud service provider i prefers some resource pooling pattern strategy in V_i to k_i. The algorithm proceeds by successively adding Static Workload preference limitations to the elastic scaling.

Keywords
Elastic scaling, measurability, Static Workload, preference limitation, resource pooling pattern, Totally Ordered Data-Intensive Systems.

1. INTRODUCTION
In an elastic scaling, it is natural to assume that a cloud service provider reasons about its cloud consumers before making a decision. Namely, in order to evaluate the possible consequences of a decision, the cloud service provider must form some resource pooling pattern conjecture about its cloud consumers' choices which, in turn, must be based on some resource pooling pattern conjecture about its cloud consumers' conjecture about their cloud consumers' choices, and so on. It is the goal of *elastic scaling* [1] to formally describe such reasoning processes, and to investigate their behavioral implications.

Throughout this paper we take a cloud service provider set perspective to analyze elastic scaling-theoretic situations. That is, we always view the elastic scaling from the perspective of cloud service provider set, and put restrictions only on the conjecture of this particular cloud service provider set – including conjecture about the cloud consumers' conjecture – without imposing restrictions on the actual conjecture of the cloud consumers. We premise this approach to be plausible; as we cannot look inside the cloud consumers at the time we make a Static Workload choice. So, can only base Static Workload choice on conjecture about the cloud consumers, and not on the actual conjecture and Static Workload choices of cloud consumers. But then, if we want to analyze the reasonable Static Workload choices a cloud service provider can make in an elastic scaling, it is sufficient to concentrate only on the conjecture of this particular cloud service provider set, as they encompass everything that can be used to make a decision. Although we premise the cloud service provider set perspective to be very natural, it crucially differs from the usual approach to elastic scaling in papers and articles, which typically proceed by imposing restrictions on the conjecture of all cloud service provider set, and not only cloud service provider set.

Measurability is a concept within elastic scaling that is based upon the following two assumptions:

- A cloud service provider should be *cautious*, that is, a cloud service provider should not exclude any cloud consumer's resource pooling pattern strategy from consideration;
- A cloud service provider should consider the cloud consumers' resource pooling pattern preferences, that is, if the cloud service provider premises that an cloud consumer prefers resource pooling pattern strategy k_i to resource pooling pattern strategy k'_i, then the cloud service provider should deem k_i much more likely k'_i.

Any resource pooling pattern strategy that can be chosen optimally under common resource pooling pattern conjecture in these two events is called *measurable*.

In order to define measurability formally we can no longer model the cloud service providers' conjecture by standard probability distributions. Suppose, for instance, that cloud service provider 1 premises that cloud service provider 2 prefers resource pooling pattern strategy *a* to resource pooling pattern strategy *b*. If cloud service provider 1's resource pooling pattern conjecture about 2's choice would be modeled by a single probability distribution then cloud service provider 1 should assign probability 0 to *b*, since it must consider 2's resource pooling pattern preferences. This, however, would contradict the assumption that it is cautious.

A possible way to define measurability is by means of *sequences of probability distributions*, or by using *totally ordered Data-intensive systems* [9], [10], [12], [14] [11], [24]. Both frameworks can model a state of mind in which you deem some cloud consumer's resource pooling pattern strategy k_i infinitely more likely than some other resource pooling pattern strategy k'_i, without completely discarding the latter choice.

The practical disadvantage of these richer frameworks is that, it makes the computation of measurable resource pooling

pattern strategies rather difficult. This is probably also the reason that measurability, despite its strong intuitive appeal, has not received as much attention as many other concepts in elastic scaling. It would therefore be very useful to have an algorithm helping us to compute this measurable resource pooling pattern strategies. A procedure, called *iteratively trembling*, that for any given $\alpha > 0$ yields the set of α-measurable resource pooling pattern strategies. By letting α tend to zero, we finally would obtain the set of measurable resource pooling pattern strategies. So, in a sense, this procedure only *indirectly* leads to the set of measurable resource pooling pattern strategies, as we first have to apply the procedure for a sequence of small α's, and then let α go to zero.

There is another algorithm designed for measurability, called *iterated backward inference.* This procedure does not exactly yield the set of measurable resource pooling pattern strategies, as its output may contain resource pooling pattern strategies that are not measurable. The output, however, always *includes* the set of measurable resource pooling pattern strategies.

In this paper we present an algorithm, called *iterated addition of Static Workload preference limitations* that *directly* delivers the set of all measurable resource pooling pattern strategies in every finite elastic scaling operation [17], [19], [23]. The algorithm is based on the new notion of a *Static Workload preference limitation*. Formally, a Static Workload preference limitation for cloud service provider i is a pair (k_i, V_i), where k_i is a resource pooling pattern strategy and V_i a subset of resource pooling pattern strategies for cloud service provider i. The interpretation is that cloud service provider i prefers some resource pooling pattern strategy V_i in to k_i, without specifying which one (unless V_i contains only one resource pooling pattern strategy, of course). A *totally ordered resource pooling pattern conjecture* for cloud service provider i about its cloud consumers' resource pooling pattern strategies is a finite sequence $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ of probability distributions on K_{-i}, the set of cloud consumers' resource pooling pattern strategy combinations, such that every resource pooling pattern strategy combination k_{-i} in K_{-i} receives positive probability under some probability distribution ψ_i^p in this sequence. For every $p \in \{1, \ldots, P\}$, we call ψ_i^p the level p resource pooling pattern conjecture.

The totally ordered resource pooling pattern conjecture ψ_i deems some resource pooling pattern strategy combination k_{-i} *infinitely more likely* than some other resource pooling pattern strategy combination k'_{-i} if there is some level p such that k_{-i} receives positive probability under the level p resource pooling pattern conjecture ψ_i^P, whereas k'_{-i} receives probability zero under the first p levels. We say that ψ_i *consider* a Static Workload preference limitation (k_j, V_j), for cloud consumer j if it deems some resource pooling pattern strategy in V_j infinitely more likely than k_j. This thus mimics the condition in measurability that i must consider j's resource pooling pattern preferences. The totally ordered resource pooling pattern conjecture ψ_i is said to *assume* a subset $E_{-i} \subseteq K_{-i}$ of resource pooling pattern strategy combinations if it deems every element in E_{-i} infinitely more likely than every element outside E_{-i}.

The algorithm we present proceeds by inductively adding Static Workload preference limitations [6], [2], [4], [3], [8], [5], [26] until no further Static Workload preference limitations can be produced. At round 1, we start with the empty set of Static Workload preference limitations for all cloud service providers. In every subsequent round, we add a Static Workload preference limitation (k_i, V_i) for cloud service provider i if every totally ordered resource pooling pattern conjecture on K_{-i} that consider all current Static Workload preference limitations for i's cloud consumers, assumes some subset $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some randomized resource pooling pattern strategy on V_i. We continue this process until no further Static Workload preference limitation can be added. Among the final set of Static Workload preference limitations for cloud service provider i, we look for those resource pooling pattern strategies k_i that are not part of any Static Workload preference limitation (k_i, V_i). We show that these resource pooling pattern strategies are exactly the measurable resource pooling pattern strategies for cloud service provider i.

So, at every round the algorithm produces, for each cloud service provider, a set of Static Workload preference limitations. As the set of Static Workload preference limitations can only grow at every round, and there are only finitely many possible Static Workload preference limitations, the algorithm must stop after finitely many rounds.

Not only can this algorithm be used to *compute* the measurable resource pooling pattern strategies in an elastic scaling, it also represents a natural *inductive reasoning procedure* for the cloud service providers that eventually lead them to measurable resource pooling pattern choices. The central object in this reasoning process is that of a Static Workload preference limitation. If we add a Static Workload preference limitation (k_i, V_i) for cloud service provider i, then normally this means that i's cloud consumers premises that i prefers some resource pooling pattern strategy in V_i to k_i. Moreover, if i's cloud consumers consider i's resource pooling pattern preferences, as we assume in measurability, then i's cloud consumers will also deem some resource pooling pattern strategy in V_i infinitely more likely than k_i. Thus, by adding Static Workload preference limitations at every round, we further and further limit the possible totally ordered conjecture that cloud service providers can plausibly hold about their cloud consumers' choices. In a sense, what the algorithm shows is that, in order to reason your way toward measurable resource pooling pattern strategies, it is sufficient to keep track of the cloud service providers' Static Workload preference limitations. At every round, by considering the current Static Workload preference limitations, we can possibly derive new Static Workload preference limitations, thus further limitations the cloud service providers' possible totally ordered conjecture, until this reasoning process cannot produce any new Static Workload preference limitations. This is where the reasoning procedure ends, and by looking at the final Static Workload preference limitations we can find the entire measurable resource pooling pattern strategies in the elastic scaling.

In the algorithm we present, the objects of output are different than in previous procedure. There, the procedure delivers at every round and for every cloud service provider i, a set of full support probability distributions on cloud service provider i's resource pooling pattern strategy, where this set becomes smaller with every round. As there are infinitely many possible sets of full support probability distributions, previous procedure can produce infinitely many possible outputs in every round. This is a major difference with the algorithm we propose, where at every round there are only a finite number

of possible outputs, namely the Static Workload preference limitations at that round. Note also that the algorithm in this paper is fundamentally different from most other inductive concepts in elastic scaling, which usually proceed by successively eliminating resource pooling pattern strategies from the elastic scaling. Think, for instance, of iterated elimination of strictly (weakly) dominated resource pooling pattern strategies. So, why did we not base the algorithm on elimination of resource pooling pattern strategies as well? The reason is that iterated elimination of strategies cannot work for measurability. In Section 2 we provide an algorithm for measurability must necessarily be of a different nature than the ones we are used to.

The outline of the paper is as follows. In Section 2 we show, why successive elimination of resource pooling pattern strategies does not work for measurability. In Section 3 we give a formal necessary and sufficient condition of measurability, by making use of totally ordered Data-intensive systems [9], [10], [12], [14] [11]. In Section 4 we present the algorithm, illustrate it by means of our main proposition showing that the algorithm produces exactly the set of measurable resource pooling pattern strategies. In Section 5 we discuss some important properties of the algorithm: We show how the algorithm can be viewed as a natural inductive reasoning procedure, and explain why the order in which we add Static Workload preference limitations does not matter for the eventual output. In section 6 we include conclusion and future scope.

2. WHY ELIMINATION OF RESOURCE POOLING PATTERN STRATEGIES DOES NOT WORK

Most algorithms in the elastic scaling literature [1], [27] proceed by successively modifying resource pooling pattern *strategies* from the operation cycle. Think, for instance, of iterated elimination of strictly (weakly) dominated resource pooling pattern strategies. As announced, the algorithm we propose for measurability is of a different nature since it is based on successively adding *Static Workload preference limitations* rather than eliminating resource pooling pattern *strategies*. A natural question is why we do not stick to the process of eliminating resource pooling pattern strategies here. In this section we show why elimination of resource pooling pattern strategies does not work for measurability.

Let us first be precise about the class of resource pooling pattern strategy elimination procedures we consider. All the elimination procedures mentioned above have in common that at each round, only weakly dominated resource pooling pattern strategies in the cloud consumer cycle of elastic scaling cycle [18], [20], [22] (but not necessarily all) are eliminated. Now, say that a resource pooling pattern strategy elimination procedure is *regular* if at every round, it eliminates a (possibly empty) subset of the set of weakly dominated resource pooling pattern strategies in the cloud consumer of elastic scaling cycle [21].

3. NECESSARY AND SUFFICIENT CONDITION OF MEASURABILITY

3.1 Totally ordered Data-intensive systems

Totally ordered Data-intensive systems have been formally introduced as a possible way to represent a decision maker's resource pooling pattern conjecture about the data-intensive state of the data-intensive world. The essential feature is that it allows the decision maker to deem one data-intensive state much more likely (in fact, infinitely more likely) than some other data-intensive state, without completely ignoring the latter data-intensive state when making a decision.

More formally, let N be some finite set of data-intensive states. By $\theta(N)$ we denote the set of all probability distributions on N. A *Totally ordered Data-intensive systems* (TODIS) on N is a finite sequence of probability distributions

$$\psi = (\psi^1, \psi^2, \ldots, \psi^P),$$

with $\psi^p \in \theta(N)$ for all $p \in \{1, \ldots, P\}$. We refer to ψ^1 as the decision maker's *level* 1 *resource pooling pattern conjecture*, to ψ^2 as its *level* 2 *resource pooling pattern conjecture*, and so on. The interpretation is that the decision maker attaches much more importance to its level 1 resource pooling pattern conjecture than to its level 2 resource pooling pattern conjecture, attaches much more importance to its level 2 resource pooling pattern conjecture than to its level 3 resource pooling pattern conjecture, and so on, without completely discarding any of these conjecture. For every data-intensive state $n \in N$, let $lp(n,\psi)$ be the first level p for which $\psi^p(n)>0$. If $\psi^p(n) = 0$ for every $p \in \{1, \ldots, P\}$, set $lp(n,\psi) = \infty$. We call $lp(n,\psi)$ the *rank* of data-intensive state n within the TODIS ψ. We say that the TODIS ψ deems data-intensive state n *infinitely more likely* than some other data-intensive state *Proposition* if n has a lower rank that *Proposition.*

3.2 Elastic scaling Planning Model

Consider a finite static elastic scaling $\delta = (K_i , x_i)_{i\in I}$ where I is the finite set of cloud service providers, the finite set K_i denotes the set of strategies for cloud service provider i, and $x_i : \prod_{j\in I} K_j \rightarrow$F denotes cloud service provider i's utility function. We assume that cloud service provider i does not only have a resource pooling pattern conjecture about its cloud consumers' resource pooling pattern strategy choices, but also about the possible conjecture that its cloud consumers could have about the other cloud service providers' resource pooling pattern strategy choices, and about the possible conjecture that the cloud consumers could have about the possible conjecture that their cloud consumers could have about the other cloud service providers' resource pooling pattern strategy choices, and so on. That is, cloud service provider i hold a full *resource pooling pattern conjecture hierarchy* about the cloud consumers' choices and the cloud consumers' conjecture. If we assume, moreover, that each of the conjecture in this hierarchy can be represented by a TODIS, this leads to the following elastic scaling planning model [7], [13], [15], [16], [25].

Necessary and sufficient condition 3.1 *(elastic scaling planning model).* A finite elastic scaling planning model for the elastic scaling δ is a tuple $(T_i, \psi_i)_{i\in I}$ where, for all cloud service providers i, T_i is a finite set of Static Workload types, and ψ_i is a function that assigns to every Static Workload type $t_i \in T_i$ some TODIS $\psi_i(t_i)$ on the set $K_{-i}\times T_{-i}$ of cloud consumers' strategy–Static Workload type combinations.

Here, $K_{-i} := \prod_{j\neq i} K_j$ denotes the set of cloud consumers' strategy combinations, and $T_{-i} := \prod_{j\neq i} T_j$ the set of cloud consumers' Static Workload type combinations. The interpretation is that $\psi_i(t_i)$ represents the resource pooling pattern conjecture that Static Workload type t_i has about its cloud consumers' choices and conjecture. For instance, the marginal of $\psi_i(t_i)$ on P_j represents the resource pooling pattern conjecture that t_i has

about cloud consumer j's choice. Since every cloud consumer's type t_j holds a resource pooling pattern conjecture about the other cloud service providers' choices, we can derive from $\psi_i(t_i)$ as well the resource pooling pattern conjecture that Static Workload type t_i has about the resource pooling pattern conjecture that cloud service provider j has about its cloud consumers' choices, and so on. In fact, from $\psi_i(t_i)$ we can derive the full resource pooling pattern conjecture hierarchy that cloud service provider i has about its cloud consumers' choices and conjecture.

The reader may wonder why we limit attention to elastic scaling planning models with *finitely* many Static Workload types for every cloud service provider. In principle, we could allow for infinitely many Static Workload types for every cloud service provider, and define measurability for such infinite elastic scaling planning models. But it can be shown that every measurable strategy in a finite elastic scaling can be supported by a measurable Static Workload type within an elastic scaling planning model with *finitely* many Static Workload types only. So, we do not "overlook" any measurable strategies by concentrating on finite Static Workload type spaces only. As working with finite sets of Static Workload types makes things easier, we have decided to solely concentrate on finite elastic scaling planning models in this paper.

Note that within an elastic scaling planning model, the totally ordered resource pooling pattern conjecture $\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ of a Static Workload type t_i is, mathematically speaking, an TODIS on the set of data-intensive states $K_{-i} \times T_{-i}$. For every cloud consumers' resource pooling pattern strategy–Static Workload type combination $(k_{-i}, t_{-i}) \in K_{-i} \times T_{-i}$, we can thus define the *rank* $lp((k_{-i}, t_{-i}), \psi_i(t_i))$ of (k_{-i}, t_{-i}) within $\psi_i(t_i)$, being the lowest level p such that $\psi_i^p(k_{-i}, t_{-i}) > 0$. Remember that, by convention, $lp((k_{-i}, t_{-i}), \psi_i(t_i)) = \infty$ whenever (k_{-i}, t_{-i}) does not receive positive probability anywhere in $\psi_i(t_i)$. We say that Static Workload type t_i deems the resource pooling pattern strategy–Static Workload type combination (k_{-i}, t_{-i}) *infinitely more likely* than some other combination (k'_{-i}, t'_{-i}) if the rank of (k_{-i}, t_{-i}) is lower than the rank of (k'_{-i}, t'_{-i}).

Similarly, we can define for every *event* $A \subseteq K_{-i} \times T_{-i}$ of cloud consumers' resource pooling pattern strategy–Static Workload type combinations the associated rank by

$$lp(A, \psi_i(t_i)) = min\{l((k_{-i}, t_{-i}), \psi_i(t_i) | (k_{-i}, t_{-i}) \in A\}.$$

Hence, the rank of A is the lowest level p such that ψ_i^p assigns positive probability to some element in A. This necessary and sufficient condition then allows us to define the rank of an *individual* cloud consumer's resource pooling pattern strategy–Static Workload type pair (k_j, t_j), simply by taking the rank of the event

$$\{k_j\} \times \prod_{p \neq i,j} K_p \times \{t_j\} \times \prod_{p \neq i,j} T_p.$$

So, we first take the marginal of the TODIS $\psi_i(t_i)$ on $K_j \times T_j$, and then take the rank of (k_j, t_j) inside this marginal TODIS. In a similar fashion, we can also define the rank of an individual cloud consumer's Static Workload type t_j, and of an individual cloud consumer's resource pooling pattern strategy k_j. As such, we can formally data-intensive state expressions like "$\psi_i(t_i)$ deems (k_j, t_j) infinitely more likely than (k'_j, t'_j) for cloud consumer j" or "$\psi_i(t_i)$ deems k_j infinitely more likely than k'_j for cloud consumer j", which means that the rank of the former is smaller than the rank of the latter.

We say that Static Workload type t_i *deems possible* some event $A \subseteq K_{-i} \times T_{-i}$ if there is some level p with $\psi_i^p(A) > 0$. That is, A is deemed possible if and only if $lp(A, \psi_i(t_i)) \neq \infty$. Since we have defined the rank also for individual resource pooling pattern strategy–Static Workload type pairs (k_j, t_j) and for individual Static Workload types t_j, we can also formally define the event that Static Workload type t_i deems possible a resource pooling pattern strategy–Static Workload type pair (k_j, t_j) for cloud consumer j, and that t_i deems possible an cloud consumer's Static Workload type t_j. It simply means that the associated rank is not ∞.

3.3 Cautious Static Workload Types

Intuitively, *caution* means that the cloud service provider should not fully exclude any cloud consumer's Static Workload choice from consideration. The formal necessary and sufficient condition is, however – in data-intensive states that a Static Workload type t_i should not exclude any strategy choice *for any cloud consumer's Static Workload type* t_j *considers possible*. Hence, for every resource pooling pattern conjecture hierarchy that t_i deems possible for its cloud consumer j, and for every measurable strategy k_j that j can possibly choose, Static Workload type t_i should deem possible the event that its cloud consumer holds this resource pooling pattern conjecture hierarchy *and* chooses k_j.

Necessary and sufficient condition 3.2 *(Cautious Static Workload type).* Consider an elastic scaling planning model with sets of Static Workload types T_i for every cloud service provider i. Static Workload type $ti \in T_i$ is cautious if, for every cloud consumer j, every Static Workload type $tj \in T_j$ it considers possible, and every resource pooling pattern strategy choice $k_j \in K_j$, Static Workload type t_i deems possible the strategy–Static Workload type pair (k_j, t_j).

3.4 Considering the cloud consumers' resource pooling pattern preferences

The key condition for measurability is that a Static Workload type should *consider its cloud consumers'* Static Workload *resource pooling pattern preferences.* In words it means that, whenever Static Workload type t_i premises that its cloud consumer j prefers some resource pooling pattern strategy k_j to some other resource pooling pattern strategy k'_j, then it should deem k_j infinitely more likely than k'_j. We must first define what it means, within our elastic scaling planning model, that a Static Workload type prefers some resource pooling pattern strategy to another resource pooling pattern strategy.

Consider a Static Workload type t_i with an TODIS $\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ on $K_{-i} \times T_{-i}$. Then, for every level $p \in \{1, \ldots, P\}$ and every resource pooling pattern strategy k_i, we can define the *level p expected utility*

$$x_i(k_i, \psi_i^p) := \sum_{(k_{-i}, t_{-i}) \in K_{-i} \times T_{-i}} \psi_i^P(k_{-i}, t_{-i})\, x_i(k_i, k_{-i}).$$

This is the expected utility that would result by choosing k_i under the resource pooling pattern conjecture ψ_i^P.

Necessary and sufficient condition 3.3 *(A Static Workload type's preference relation over* resource pooling pattern *strategies).* Let $t_i \in T_i$ be a Static Workload type with TODIS

$\psi_i(t_i) = (\psi_i^1, \ldots, \psi_i^P)$ on $K_{-i} \times T_{-i}$. Static Workload type t_i prefers resource pooling pattern strategy k_i to some other resource pooling pattern strategy k'_i if there is some level $p \in \{1, \ldots, P\}$ such that $x_i(k_i, \psi_i^o) > x_i(k'_i, \psi_i^o)$ and $x_i(k_i, \psi_i^o) = x_i(k'_i, \psi_i^o)$ for all $o<p$.

For later purposes, we say that Static Workload type t_i *weakly prefers* k_i *to* k'_i if t_i does not prefer k'_i to k_i.

Necessary and sufficient condition 3.4 *(Considering the cloud consumers' resource pooling pattern preferences).* Let $t_i \in T_i$ be a cautious Static Workload type. Static Workload type t_i consider the cloud consumer's resource pooling pattern preferences if, for every cloud consumer j, every Static Workload type $t_j \in T_j$ deemed possible by t_i, and every two strategies k_j, k'_j such that t_j prefers k_j to k'_j, Static Workload type t_i deems the pair (k_j, t_j) infinitely more likely than the pair (k'_j, t_j).

3.5 Measurability

We say that a Static Workload type t_i is *measurable* if t_i is cautious and consider the cloud consumers' resource pooling pattern preferences, premises that all cloud consumers are cautious and consider their cloud consumers' resource pooling pattern preferences, premises that all cloud consumers premise that their cloud consumers are cautious and consider their cloud consumers' resource pooling pattern preferences, and so on. In other words, t_i is cautious and consider the cloud consumers' resource pooling pattern preferences, and expresses common resource pooling pattern conjecture in the event that cloud service providers are cautious and consider the cloud consumers' resource pooling pattern preferences.

Necessary and sufficient condition 3.5 *(Common resource pooling pattern conjecture in "caution and consider of the cloud consumers' resource pooling pattern preferences").* A Static Workload type t_i expresses common resource pooling pattern conjecture in the event that cloud service providers are cautious and consider the cloud consumers' resource pooling pattern preferences if t_i only deems possible cloud consumers' Static Workload types that are cautious and consider their cloud consumers' resource pooling pattern preferences, only deems possible cloud consumers' Static Workload types that only deem possible cloud consumers' Static Workload types that are cautious and consider their cloud consumers' resource pooling pattern preferences, and so on.

By additionally assuming that t_i itself is cautious and consider the cloud consumers' resource pooling pattern preferences, we obtain the necessary and sufficient condition of a measurable Static Workload type.

Necessary and sufficient condition 3.6 *(measurable Static Workload type).* A Static Workload type t_i is measurable if it is cautious and consider the cloud consumers' resource pooling pattern preferences, and moreover expresses common resource pooling pattern conjecture in the event that cloud service providers are cautious and consider the cloud consumers' resource pooling pattern preferences.

Finally, we say that a resource pooling pattern strategy k_i is measurable for cloud service provider i if it is optimal for some measurable Static Workload type. Formally, a resource pooling pattern strategy k_i is called *optimal* for Static Workload type t_i if t_i weakly prefers k_i to any other resource pooling pattern strategy.

Necessary and sufficient condition 3.7 *(measurable* resource pooling pattern *strategy).* A resource pooling pattern strategy k_i for cloud service provider i is measurable if there is some finite elastic scaling planning model $(T_i, \psi_i)_{i \in I}$ and some measurable Static Workload type $t_i \in T_i$ such that k_i is optimal for t_i.

As we already mentioned before, the concept of a measurable resource pooling pattern strategy would not change if we would allow for infinite elastic scaling planning models here.

4. ALGORITHM

In this section we will present an algorithm that always delivers all measurable resource pooling pattern strategies. Before doing so, we first provide some intuitive arguments that eventually will lead to the algorithm. Finally, we state our main result, namely that the algorithm yields precisely the set of measurable resource pooling pattern strategies in every elastic scaling.

4.1 Road to the Algorithm

In Section II we have seen that elimination of (subsets of) weakly dominated resource pooling pattern strategies cannot work for measurability. So, what kind of procedure *could* work here? We start our informal investigation with the following well-known fact:

Step 1. Suppose that resource pooling pattern strategy k_i is weakly dominated on K_{-i} by some randomized resource pooling pattern strategy $\gamma_i \in \theta(V_i)$, where V_i is a subset of resource pooling pattern strategies. Then, if cloud service provider i is cautious, it will prefer some resource pooling pattern strategy in V_i to k_i. We say that (k_i, V_i) is a resource pooling pattern *Static Workload preference limitation* for cloud service provider i.

Here, $\theta(V_i)$ denotes the set of probability distributions on V_i. The reason for this fact is simple: If k_i is weakly dominated by resource pooling pattern γ_i, then under every cautious totally ordered resource pooling pattern conjecture, k_i will be worse than γ_i, and hence there must be some $v_i \in V_i$ which is better than k_i under such a cautious totally ordered resource pooling pattern conjecture. So, (k_i, V_i) will be a resource pooling pattern Static Workload preference limitation for cloud service provider i.

Suppose now that cloud service provider i premise its cloud consumers are cautious and that it consider its cloud consumers' resource pooling pattern preferences. If some cloud consumer's resource pooling pattern strategy k_j is weakly dominated on K_{-j} by some randomized resource pooling pattern strategy $\gamma_i \in \theta(V_i)$, then we know by Step 1 that cloud service provider j will prefer some resource pooling pattern strategy in V_j to k_j in case it is cautious. As cloud service provider i indeed premises it is cautious, and consider j's resource pooling pattern preferences, cloud service provider i must deem some resource pooling pattern strategy in V_j infinitely more likely than k_j. We say that cloud service provider i's totally ordered resource pooling pattern conjecture consider the preference limitation (k_j, V_j). This leads to the following observation:

Step 2. Suppose cloud service provider i premises its cloud consumers are cautious, and consider its cloud consumers' resource pooling pattern preferences. Then, i's totally ordered resource pooling pattern conjecture must consider every cloud

consumer's resource pooling pattern Static Workload preference limitation (k_j, V_j) generated in Step 1.

Say that a totally ordered resource pooling pattern conjecture for cloud service provider i *assumes* a set $E_{-i} \subseteq K_{-i}$ of cloud consumers' resource pooling pattern strategy combinations if it deems all resource pooling pattern strategy combinations inside E_{-i} infinitely more likely than all resource pooling pattern strategy combinations outside E_{-i}. Suppose now that i's totally ordered resource pooling pattern conjecture is cautious, and assumes some set E_{-i} of cloud consumers' resource pooling pattern strategy combinations. Assume, moreover, that its resource pooling pattern strategy k_i is weakly dominated on E_{-i} by a randomized resource pooling pattern strategy $\gamma_i \in \theta(V_i)$. Then, i must prefer some resource pooling pattern strategy in V_i to k_i. The argument is basically the same as for Step 1, if we would "reduce" the elastic scaling to cloud consumers' resource pooling pattern strategy combinations in E_{-i}. We thus obtain the following step:

Step 3. Suppose that every totally ordered resource pooling pattern conjecture for cloud service provider i considering all Static Workload preference limitations from Step 1, assumes some $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. Suppose, moreover, that cloud service provider i is cautious, premises its cloud consumers are cautious, and consider the cloud consumers' resource pooling pattern preferences. Then, i must prefer some resource pooling pattern strategy in V_i to k_i. We say that (k_i, V_i) *is* a *new Static Workload preference limitation* for cloud service provider i.

Of course, we can iterate this argument if we assume that cloud service provider i is cautious, consider the cloud consumers' resource pooling pattern preferences, and expresses common resource pooling pattern conjecture in the event that cloud service providers are cautious and consider the cloud consumers' resource pooling pattern preferences. That is, if we assume that cloud service provider i's Static Workload type is measurable. The inductive step would then look as follows:

Inductive step. Suppose that every totally ordered resource pooling pattern conjecture for i that consider all Static Workload preference limitations generated so far, assumes some $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. Then, if i is of a measurable Static Workload type, it must prefer some resource pooling pattern strategy in V_i to k_i. So, (k_i, V_i) would be a new Static Workload preference limitation for cloud service provider i.

This would thus generate an inductive procedure in which at every step (possibly) some new Static Workload preference limitations would be added for the cloud service providers. Since there are only finitely many possible Static Workload preference limitations for the cloud service providers, this procedure must end after finitely many steps. Now, consider some cloud service provider i, and its set of Static Workload preference limitations generated by the procedure above.

If cloud service provider i is of some measurable Static Workload type, we know from our arguments above that it will never choose a resource pooling pattern strategy k_i if it is part of some Static Workload preference limitation (k_i, V_i). In that case, namely, it would always prefer some resource pooling pattern strategy in V_i to k_i, so k_i could not be optimal.

So, the procedure above rules out resource pooling pattern strategies that is certainly not measurable. But what about the converse? So, what about resource pooling pattern strategies that are not ruled out by the procedure above? The main proposition in this paper, Proposition 4.6, will show that the "surviving" resource pooling pattern strategies are all measurable! Hence, the procedure above will always select *exactly* those resource pooling pattern strategies that are measurable – not more and not less.

4.2 Description of the algorithm

Before we state the algorithm, we first formally necessary and sufficient condition the new concepts we described above, such as Static Workload preference limitations, what it means for a totally ordered resource pooling pattern conjecture to consider a Static Workload preference limitation, and so on.

Necessary and sufficient condition 4.1 *(Static Workload preference limitation).* A Static Workload preference limitation for cloud service provider i is a pair (k_i, V_i) where k_i is a resource pooling pattern strategy, and V_i a nonempty subset of resource pooling pattern strategies.

The interpretation is that cloud service provider i prefers at least one resource pooling pattern strategy from V_i to k_i. Now, consider a totally ordered resource pooling pattern conjecture ψ_i on K_{-i}, which is simply a TODIS on K_{-i}. From here on, we will always assume that such a totally ordered resource pooling pattern conjecture ψ_i has full support on K_{-i}, that is, every resource pooling pattern strategy combination in K_{-i} receives positive probability in some level of ψ_i.

Necessary and sufficient condition 4.2 *(Considering a Static Workload preference limitation).*A totally ordered resource pooling pattern conjecture ψ_i on K_{-i} consider a Static Workload preference limitation (k_j, V_j) *for* cloud service provider j if ψ_i deems some resource pooling pattern strategy in V_j infinitely more likely than k_j.

This, in a sense, mimics the requirement that cloud service provider i must consider j's resource pooling pattern preferences.

Necessary and sufficient condition 4.3 *(Assuming a set of cloud consumers' resource pooling pattern strategy combinations).* Consider a subset $E_{-i} \subseteq K_{-i}$ of cloud consumers' resource pooling pattern strategy combinations, and a totally ordered resource pooling pattern conjecture ψ_i on K_{-i}. The totally ordered resource pooling pattern conjecture ψ_i assumes the set E_{-i} if ψ_i deems all resource pooling pattern strategy combinations inside E_{-i} infinitely more likely than all resource pooling pattern strategy combinations outside E_{-i}.

Note that a totally ordered resource pooling pattern conjecture $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ on K_{-i} assumes a subset $E_{-i} \subseteq K_{-i}$ if and only if, there is some level $p \in \{1, \ldots, P\}$ such that $\bigcup_{o \leq p} \text{supp}(\ ) = E_{-i}$. Here, $\text{supp}(\psi_i^o)$denotes the support of the probability distribution ψ_i^o. A *randomized resource pooling pattern strategy* for cloud service provider i is a probability distribution $\gamma_i \in \theta(K_i)$.on cloud service provider i's resource pooling pattern strategies. For a subset $V_i \subseteq K_i$, we denote by $\theta(V_i)$ the set of randomized resource pooling pattern strategies that assign positive probability only to resource pooling pattern strategies in V_i. For some cloud consumers' resource pooling pattern strategy combination $k_{-i} \in K_{-i}$, let

$$x_i(\gamma_i, k_{-i}) := \sum \gamma_i(k_i)\, x_i(k_i, k_{-i})$$

$k_i \in K_i$

denote i's expected utility from the randomized resource pooling pattern strategy γ_i and the cloud consumers' resource pooling pattern strategy combination k_{-i}.

Necessary and sufficient condition 4.4 *(Weakly dominated resource pooling pattern strategy).* Let $E_{-i} \subseteq K_{-i}$ be a subset of the cloud consumers' resource pooling pattern strategy combinations. Resource pooling pattern Strategy k_i is said to be weakly dominated by randomized resource pooling pattern strategy γ_i on E_{-i} if $x_i(\gamma_i , k_{-i}) \geq x_i(k_i , k_{-i})$ for all $k_{-i} \in E_{-i}$, with strict in equality for at least some $k_{-i} \in E_{-i}$.

We are now ready to present the algorithm. The idea is to start with the empty set of Static Workload preference limitations for all cloud service providers, and at every round to add new Static Workload preference limitations, if possible. For that reason, the algorithm is called "iterated addition of Static Workload preference limitations".

Algorithm 4.5 *(Iterated addition of Static Workload preference limitations).* In round 1, begin for all cloud service providers i with the empty set of Static Workload preference limitations.

At every further round $q \geq 2$, limit for every cloud service provider i to those totally ordered resource pooling pattern conjecture on K_{-i} that consider all cloud consumers' Static Workload preference limitations generated so far. Add a new Static Workload preference limitation (k_i, V_i) for cloud service provider i if every such totally ordered resource pooling pattern conjecture assumes some set $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

Since the number of Static Workload preference limitations is finite, this algorithm must end after a finite number of rounds. We say that resource pooling pattern strategy k_i *survives* the algorithm of iterated addition of Static Workload preference limitations if k_i is not part of any Static Workload preference limitation (k_i, V_i) generated by the algorithm. Namely, if k_i were to be part of a Static Workload preference limitation (k_i, V_i) produced by the algorithm, then cloud service provider i would prefer at least one strategy in V_i to k_i, and hence k_i could not be optimal.

4.3 Main Proposition

Our main proposition states that the algorithm of iterated addition of Static Workload preference limitations yields *exactly* the set of measurable resource pooling pattern strategies for every cloud service provider.

Proposition 4.6 *(Algorithm yields precisely the set of measurable resource pooling pattern strategies). Consider a finite static elastic scaling. Then, a resource pooling pattern strategy k_i is measurable, if and only if, k_i survives the algorithm of iterated addition of Static Workload preference limitations.*

The easier direction is to show that every measurable resource pooling pattern strategy survives iterated addition of Static Workload preference limitation. So, a measurable resource pooling pattern strategy k_i can never be part of a Static Workload preference limitation (k_i, V_i) generated by the algorithm. The more difficult direction is to prove that every resource pooling pattern strategy k_i that is not part of any such Static Workload preference limitation (k_i, V_i) is measurable. Hence, we must construct an elastic scaling planning model in which each of this resource pooling pattern strategies k_i is supported by some measurable Static Workload type. This construction is rather delicate.

From the proposition, we can easily derive the following observation: If in a given elastic scaling *no* resource pooling pattern strategy is weakly dominated, then *all* resource pooling pattern strategies for the cloud service providers are measurable. Namely, the algorithm we present will only generate Static Workload preference limitations at the first round if there is at least some resource pooling pattern strategy that is weakly dominated within the full elastic scaling. Otherwise, the algorithm will not generate any Static Workload preference limitation at all, and hence all resource pooling pattern strategies would survive the algorithm.

4.4 A Finite Formulation of the Algorithm

The algorithm of iterated addition of Static Workload preference limitations as we have formulated it proceeds by adding Static Workload preference limitations and deleting totally ordered conjecture at every round. More precisely, we start with the empty set of Static Workload preference limitations and the full set of totally ordered conjecture. At the first round we see whether we can add some Static Workload preference limitations. If so, then this would reduce the set of totally ordered conjecture, which at the next round could add some further Static Workload preference limitations, and so on.

What is somewhat undesirable from a computational point of view is that there are infinitely many possible totally ordered conjecture in the elastic scaling. This would suggest that at every round in the algorithm we must scan through infinitely many totally ordered conjecture. This, however, is not necessary. What matters for the algorithm is not so much the precise probabilities in the totally ordered resource pooling pattern conjecture, but the induced "likelihood resource pooling pattern ordering" on cloud consumers' resource pooling pattern strategy combinations. More precisely, let $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$ be a totally ordered resource pooling pattern conjecture on K_{-i}. Remember our convention that ψ_i has full support on K_{-i}, that is, every $k_{-i} \in K_{-i}$ receives positive probability in some level ψ_i^p. Let $O_i = (O_i^1, \ldots, O_i^Z)$ be the ordered sequence of disjoint subsets $O_i^z \subseteq K_{-i}$ such that (a) ψ_i deems every $k_{-i} \in O_i^z$ infinitely more likely than every $K_{-i} \in O_i^{z+1}$ for every $z \in \{1, \ldots, Z-1\}$, (b) for every m and every $k'_{-i}, k_{-i} \in O_i^z$, the TODIS ψ_i does not deem k_{-i} infinitely more likely than k'_{-i}, nor vice versa, and (c) the union of the sets in O_i is K_{-i}. We call O_i the *likelihood ordering* induced by ψ_i. Formally, we have the following necessary and sufficient condition.

Necessary and sufficient condition 4.7 *(Likelihood ordering).* A likelihood ordering for cloud service provider i on the cloud consumers' resource pooling pattern strategy combinations is an ordered sequence $O_i = (O_i^1, \ldots, O_i^Z)$ where $O_i^1, \ldots, O_i^Z$ are pair-wise disjoint subsets of K_{-i} whose union is equal to K_{-i}.

So, the interpretation is that O_i deems all resource pooling pattern strategy combinations in O_i^1 infinitely more likely than all resource pooling pattern strategy combinations in O_i^2, deems all resource pooling pattern strategy combinations in O_i^2 infinitely more likely than all resource pooling pattern strategy combinations in O_i^3, and so on. It is clear that there are only finitely many likelihood orderings in the elastic

scaling, since there are only finitely many resource pooling pattern strategies for every cloud service provider.

We can now easily extend the necessary and sufficient condition of "considering a Static Workload preference limitation" and "assuming a set of cloud consumers' resource pooling pattern strategy combinations" to likelihood orderings. Say that a likelihood resource pooling pattern ordering $O_i = (O_i^1, \ldots, O_i^Z)$ *consider a Static Workload preference limitation* (k_j, V_j) if O_i deems some resource pooling pattern strategy in V_j infinitely more likely than k_j. Also, the likelihood ordering O_i is said to *assume the set* E_{-i} *of cloud consumers'* resource pooling pattern *strategy combinations* if O_i deems all resource pooling pattern strategy combinations inside E_{-i}, infinitely more likely than all resource pooling pattern strategy combinations outside E_{-i}. The algorithm of iterated addition of Static Workload preference limitations can thus alternatively be stated as follows:

Algorithm 4.8 *(Finite version).* In round 1, begin for all cloud service providers i with the empty set of Static Workload preference limitations.

At every further round $q \geq 2$, limit for every cloud service provider i to those likelihood resource pooling pattern orderings on K_{-i} that consider all cloud consumers' Static Workload preference limitations generated so far. Add a new Static Workload preference limitation (k_i, V_i) for cloud service provider i if every such likelihood resource pooling pattern ordering assumes some set $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

The advantage of this formulation is that at every round, we only have to scan through finitely many objects, as there are only finitely many Static Workload preference limitations and likelihood resource pooling pattern orderings in the elastic scaling. Obviously, this algorithm generates precisely the same set of Static Workload preference limitations as the original procedure. As such, the measurable resource pooling pattern strategies are precisely those resource pooling pattern strategies that survive this alternative algorithm.

5. DISCUSSION

In this section we will discuss some important properties of the algorithm.

5.1 Algorithm as an inductive reasoning procedure

The algorithm is not merely a tool to compute the measurable resource pooling pattern strategies in an elastic scaling, but can also be interpreted as an inductive reasoning process that can be used by a cloud service provider who reasons in the spirit of measurability. Consider namely a fixed cloud service provider in the elastic scaling, say cloud service provider i. In round 2, the algorithm would add for every cloud consumer j a Static Workload preference limitation (k_j, V_j) if k_j would be weakly dominated on K_{-j} by a mixture on V_j. In that case, cloud service provider i would store the Static Workload preference limitation (k_j, V_j) in its mind, meaning that he premises that cloud service provider j prefers some resource pooling pattern strategy in V_j to k_j. If i consider j's resource pooling pattern preferences, then it should consequently deem some resource pooling pattern strategy in V_j infinitely more likely than k_j. That is, the Static Workload preference limitations that cloud service provider i would store in its mind at round 2 would limit the possible totally ordered conjecture it could hold about its cloud consumers' choices. Moreover, if cloud service provider i premises that its cloud consumers reason similarly, then cloud service provider i can actually deduce the possible totally ordered conjecture that its cloud consumers may hold at this round.

In the next round of its reasoning procedure, cloud service provider i would then ask for every cloud consumer j: Given its limited set of conjecture, would cloud service provider j always assume some set $E_{-j} \subseteq K_{-j}$ on which some resource pooling pattern strategy k_j would always be weakly dominated by a mixture on V_j? If yes, then cloud service provider i will store (k_j, V_j) as a new Static Workload preference limitation in its mind. By doing so, cloud service provider i would then further limit the possible totally ordered conjecture it could hold about its cloud consumers. Cloud service provider i could continue this inductive reasoning procedure until no new Static Workload preference limitation could be added, and hence its possible totally ordered conjecture could not be limited any further.

So we see that the algorithm may serve very well as an intuitive reasoning procedure for cloud service providers that will eventually lead them to the measurable resource pooling pattern strategies in the elastic scaling. What is crucial in this reasoning procedure is that a cloud service provider only needs to keep track of Static Workload preference limitations, which substantially simplifies matters compared to the original necessary and sufficient condition of measurability. In that light, our main proposition thus says that in order to find the measurable resource pooling pattern strategies in an elastic scaling, it is sufficient for a cloud service provider to think in terms of Static Workload preference limitations, and to reason in accordance with the algorithm.

In the elastic scaling literature [1], there are other algorithms that can nicely be interpreted as intuitive reasoning procedures. Take, for instance, the concept of *common resource pooling pattern conjecture in measurability* and the associated algorithm of *iterated elimination of strictly dominated resource pooling pattern strategies.* Here, the algorithm can be seen as a reasoning procedure in which a cloud service provider successively deletes cloud consumers' resource pooling pattern strategies, since they can no longer be optimal. At every round, this would then limit the cloud service provider's possible conjecture as it must assign probability zero to these resource pooling pattern strategies. These additional limitations on the cloud service providers' conjecture could then induce further resource pooling pattern strategies that can be deleted, and so on. So, in that procedure the cloud service providers' possible (non-totally ordered) conjecture are limited further and further by deleting resource pooling pattern strategies, whereas in our procedure the (totally ordered) conjecture are limited further and further by adding new Static Workload preference limitations.

A similar story can be told for the concept of *iterated assumption of measurability within a complete Static Workload type structure* and the associated algorithm of *iterated elimination of weakly dominated resource pooling pattern strategies.* Here, the algorithm reflects a reasoning procedure in which a cloud service provider with totally ordered conjecture iteratedly deletes weakly dominated resource pooling pattern strategies from its mind. At every

round of this procedure, the cloud service provider will then deem all surviving resource pooling pattern strategies infinitely more likely than all deleted resource pooling pattern strategies, thus limiting the possible totally ordered conjecture it can hold. So also in this procedure, the cloud service provider's possible conjecture is limited in every round by deleting resource pooling pattern strategies.

5.2 Order Independence

For the algorithm, it can be shown that the order and speed in which we add preference restrictions does not matter for the eventual result. That is, it does not matter whether in every round we add *all* preference restrictions that can possibly be generated, or only *some* of these.

To see this, let us compare two procedures, Procedure 1 and Procedure 2, where in the first we always add *all* possible Static Workload preference limitations at every round, and in the second we only add *some* of the possible Static Workload preference limitations every time. Then, first of all, Procedure 1 will at every round generate at least as many Static Workload preference limitations as Procedure 2. Namely, at round 2 Procedure 1 generates as least as many Static Workload preference limitations, by necessary and sufficient condition. Therefore, at round 3 Procedure 1 limits to a smaller set of totally ordered conjecture than Procedure 2. But then, under Procedure 1 it will be "easier" to generate new Static Workload preference limitations at round 3 than under Procedure 2. Hence, at round 3 Procedure 1 will, again, generate at least as many Static Workload preference limitations as Procedure 2, and so on. So, eventually, Procedure 1 will generate at least as many Static Workload preference limitations as Procedure 2. The key argument here was that a larger set of Static Workload preference limitations will lead to a smaller set of possible totally ordered conjecture, and a smaller set of possible totally ordered conjecture will in turn lead to a larger set of induced Static Workload preference limitations. So, the algorithm is *monotone* in this sense.

On the other hand, it can also be shown that every Static Workload preference limitation generated by Procedure 1 will also eventually be generated by Procedure 2. Suppose, namely, that Procedure 1 would generate some Static Workload preference limitation that would not be generated at all by Procedure 2. Then, let p be the first round at which Procedure 1 would generate a Static Workload preference limitation, say (k_i, V_i), not generated by Procedure 2 at all. By construction of the algorithm, every totally ordered resource pooling pattern conjecture for cloud service provider i that consider all Static Workload preference limitations generated by Procedure 1 *before* round p, must assume some set E_{-i} on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$. By our assumption, all these Static Workload preference limitations generated by Procedure 1 before round p are also eventually generated by Procedure 2, let us say before round $z \geq p$. But then, every totally ordered resource pooling pattern conjecture for cloud service provider i that consider all Static Workload preference limitations generated by Procedure 2 before round z, assumes a set E_{-i} on which k_i is weakly dominated by some $\gamma_i \in \theta(V_i)$.

Hence, Procedure 2 must add the Static Workload preference limitation (k_i, V_i) sooner or later, which is a contradiction since we assumed that Procedure 2 does not generate Static Workload preference limitation (k_i, V_i) at all. We thus conclude that every Static Workload preference limitation added by Procedure 1 is also finally added by Procedure 2. As such, Procedures 1 and 2 eventually generate exactly the same set of Static Workload preference limitations. So, indeed, the order and speed in which we add Static Workload preference limitations is irrelevant to the algorithm.

6. CONCLUSION AND FUTURE SCOPE

In this section we conclude, stating that the algorithm of iterated addition of Static Workload preference limitations selects exactly the set of measurable resource pooling pattern strategies in the elastic scaling. For our conclusion, we recall the necessary and sufficient condition of a *likelihood resource pooling pattern ordering induced by a TODIS.* Consider a TODIS $\psi_i = (\psi_i^1, \ldots, \psi_i^P)$on K_{-i}. Remember our convention that ψ_i has full support on K_{-i}, that is, every $k_{-i} \in K_{-i}$ receives positive probability in some level ψ_i^p. Let $O_i = (O_i^1, \ldots, O_i^Z)$be the ordered sequence of disjoint subsets $O_i^z \subseteq K_{-i}$ such that (a) ψ_i deems every $k_{-i} \in O_i^z$ infinitely more likely than every $k'_{-i} \in O_i^{z+1}$, for every $z \in \{1, \ldots, Z-1\}$,(b) for every z and every k_{-i} , $k'_{-i} \in O_i^z$, the TODIS ψ_i does not deem k_{-i} infinitely more likely than k'_{-i}, nor vice versa, and(c) the union of the sets in O_i is K_{-i}. We call O_i the *likelihood resource pooling pattern ordering* induced by ψ_i. Our conclusion characterizes, for a given resource pooling pattern strategy k_i and set $V_i \subseteq K_i$, those likelihood resource pooling pattern orderings on K_{-i} that admit an TODIS under which k_i is weakly preferred to all resource pooling pattern strategies in V_i. Despite the progress on our interpretation is made the following three open problems are available for further research.

Open Problem 6.1 *Let ψ_i be a TODIS on K_{-i}, let k_i be a resource pooling pattern strategy and $V_i \subseteq K_i$ a subset of resource pooling pattern strategies.*(a) *If under the TODIS ψ_i, resource pooling pattern strategy k_i is weakly preferred to all resource pooling pattern strategies in V_i. Does ψ_i assume any $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by a mixture on V_i?* (b) *If ψ_i does not assume any $E_{-i} \subseteq K_{-i}$ on which k_i is weakly dominated by a mixture on V_i. Does some TODIS ϕ_i, inducing the same likelihood resource pooling pattern ordering as ψ_i, under which k_i is weakly preferred to all resource pooling pattern strategies in V_i?*

Open Problem 6.2 *Let t_i be a measurable Static Workload type. Does t_i's totally ordered resource pooling pattern conjecture on K_{-i} consider every Static Workload preference limitation in F^{∞}_{-i}?*

Open Problem 6.3 *(Property of Static Workload preference limitations not generated by the algorithm). For every cloud service provider i, let F_i^{not} be the set of Static Workload preference limitations not generated by the algorithm. Does for every $(k_i, V_i) \in F_i^{not}$ is there an TODIS ψ_i on K_{-i} such that* (1) *under ψ_i, resource pooling pattern strategy k_i is weakly preferred to all resource pooling pattern strategies in V_i, and* (2) *for every cloud consumer's resource pooling pattern strategy k_j, the pair $((k_j, V_j^-(k_j, \psi_i))$ is in F_j^{not}?*

7. REFERENCES

[1] Kiran M., Saikat Mukherjee, Ravi Prakash G., Characterization of Randomized Shuffle and Sort Quantifiability in MapReduce Model, International Journal of Computer Applications, 51-58, Volume 79, No. 5, October 2013.

[2] Amresh Kumar, Kiran M., Saikat Mukherjee, Ravi Prakash G., Verification and Validation of MapReduce Program model for Parallel K-Means algorithm on Hadoop Cluster, International Journal of Computer Applications, 48-55, Volume 72, No. 8, June 2013.

[3] Barroso, L.A., Ho¨lzle, U.: The datacenter as a computer: an introduction to the design of warehouse-scale machines. Synth. Lect. Comput. Architect. 4, 1–45 (2009).

[4] Kiran M., Amresh Kumar, Saikat Mukherjee, Ravi Prakash G., Verification and Validation of MapReduce Program Model for Parallel Support Vector Machine Algorithm on Hadoop Cluster, International Journal of Computer Science Issues, 317-325, Vol. 10, Issue 3, No. 1, May 2013.

[5] Ravi Prakash G, Kiran M. Saikat Mukherjee, On Randomized Preference Limitation Protocol for Quantifiable Shuffle and Sort Behavioral Implications in MapReduce Programming Model, Parallel & Cloud Computing, Vol. 3, Issue 1, 1-14, January 2014.

[6] Fehling, C., Leymann, F., Mietzner, R., Schupeck, W.: A collection of patterns for cloud types, cloud service models, and cloud-based application architectures. Technical report, University of Stuttgart (2011)

[7] Ravi Prakash G, Kiran M, On The Least Economical MapReduce Sets for Summarization Expressions, International Journal of Computer Applications, 13-20, Volume 94, No.7, May 2014.

[8] Ravi (Ravinder) Prakash G, Kiran M., On Randomized Minimal MapReduce Sets for Filtering Expressions, International Journal of Computer Applications, Volume 98, No. 3, Pages 1-8, July 2014.

[9] Fehling, C., Leymann, F., Retter, R., Schumm, D., Schupeck, W.: An architectural pattern language of cloud-based applications. In: Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP), Portland, (2011).

[10] Fehling, C., Leymann, F., Rutschlin, J., Schumm, D.: Pattern-based development and management of cloud applications. Future Internet 4, 110–141 (2012). (doi:10.3390/fi4010110)

[11] Ravi (Ravinder) Prakash G, Kiran M., How Minimal are MapReduce Arrangements for Binning Expressions. International Journal of Computer Applications Volume 99 (11): 7-14, August 2014.

[12] Ravi (Ravinder) Prakash G, Kiran M., Shuffling Expressions with MapReduce Arrangements and the Role of Binary Path Symmetry. International Journal of Computer Applications 102(16): 19-24, September 2014.

[13] Dimitri P. Bertsekas and John N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Athena Scientific, Hardcover Edition (appeared in 2015), ISBN: 1-886529-15-9 Publication: 2015, 735 pages.

[14] Ravi (Ravinder) Prakash G, Kiran M; How Replicated Join Expressions Equal Map Phase or Reduce Phase in a MapReduce Structure? International Journal of Computer Applications, Volume 107 (12): 43-50, December 2014.

[15] Fehling, C., Ewald, T., Leymann, F., Pauly, M., Ru¨tschlin, J., Schumm, D.: Capturing cloud computing knowledge and experience in patterns. In: Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD), Honolulu, (2012).

[16] Bauer, E., Adams, R.: Reliability and Availability of Cloud Computing. Wiley-IEEE Press, Hoboken (2012).

[17] Ravi (Ravinder) Prakash G, Kiran M., On Composite Join Expressions of Map-side with many Reduce Phase. International Journal of Computer Applications Volume 110(9): 37-44, January 2015.

[18] Dimitri P. Bertsekas, Convex Optimization Algorithms, Athena Scientific, Hardcover Edition ISBN: 1-886529-28-0, 978-1-886529-28-1, Publication: February, 2015, 576 pages.

[19] Ravi (Ravinder) Prakash G, Kiran M; How Reduce Side Join Part File Expressions Equal MapReduce Structure into Task Consequences, Performance? International Journal of Computer Applications, Volume 105(2):8-15, November 2014

[20] Ravi (Ravinder) Prakash G, Kiran M. "On the MapReduce Arrangements of Cartesian product Specific Expressions". International Journal of Computer Applications 112(9):34-41, February 2015.

[21] Ravi (Ravinder) Prakash G, Kiran M., On Job Chaining MapReduce Meta Expressions of Mapping and Reducing Entropy Densities. International Journal of Computer Applications 113(15): 20-27, March 2015.

[22] Ravi (Ravinder) Prakash G, Kiran M. "On Chain Folding Problems of Chain Mapper and Chain Reducer Meta Expressions". International Journal of Computer Applications 116(16): 35-42, April 2015.

[23] Ravi (Ravinder) Prakash G, Kiran M."On Job Merging MapReduce Meta Expressions for Multiple Decomposition Mapping and Reducing". International Journal of Computer Applications 118 (13):14-21, May 2015.

[24] Ravi (Ravinder) Prakash G, Kiran M." Characterization of Randomized External Source Output Map Reduce Expressions". International Journal of Computer Applications 123(14):9-16, August 2015.

[25] Ravi (Ravinder) Prakash G, Kiran M., Does there Exist Pruning Decomposition for MapReduce Expressions Arrangements?. International Journal of Computer Applications 125(12): 41-48, September 2015.

[26] Ravi (Ravinder) Prakash G, Kiran M: Can one find External Source Input Expressions for which there exist Map Reduce Configurations? International Journal of Computer Applications 128(12): 14-21, October 2015.

[27] Ravi (Ravinder) Prakash G. and Kiran M. Is It True for Static Scaling Cloud Model there Exists a Centrally Asymmetric Static Workload Pattern?. Communications on Applied Electronics 3(4):39-48, November 2015.