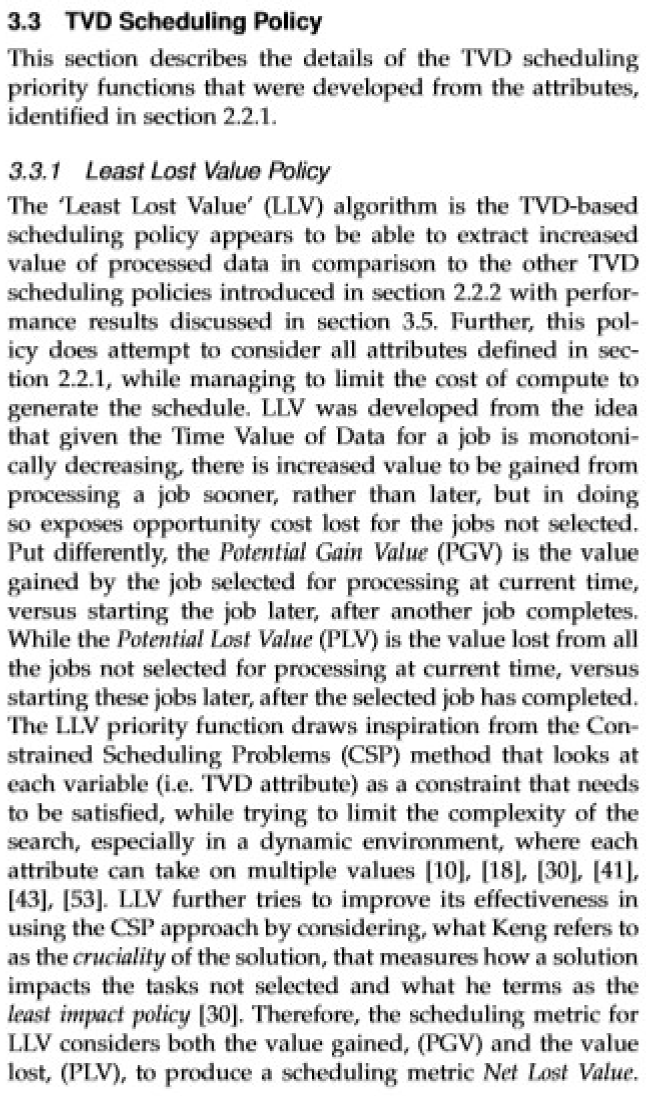


3.3 TVD Scheduling Policy

This section describes the details of the TVD scheduling priority functions that were developed from the attributes, identified in section 2.2.1.

3.3.1 Least Lost Value Policy

The 'Least Lost Value' (LLV) algorithm is the TVD-based scheduling policy appears to be able to extract increased value of processed data in comparison to the other TVD scheduling policies introduced in section 2.2.2 with performance results discussed in section 3.5. Further, this policy does attempt to consider all attributes defined in section 2.2.1, while managing to limit the cost of compute to generate the schedule. LLV was developed from the idea that given the Time Value of Data for a job is monotonically decreasing, there is increased value to be gained from processing a job sooner, rather than later, but in doing so exposes opportunity cost lost for the jobs not selected. Put differently, the *Potential Gain Value* (PGV) is the value gained by the job selected for processing at current time, versus starting the job later, after another job completes. While the *Potential Lost Value* (PLV) is the value lost from all the jobs not selected for processing at current time, versus starting these jobs later, after the selected job has completed. The LLV priority function draws inspiration from the Constrained Scheduling Problems (CSP) method that looks at each variable (i.e. TVD attribute) as a constraint that needs to be satisfied, while trying to limit the complexity of the search, especially in a dynamic environment, where each attribute can take on multiple values [10], [18], [30], [41], [43], [53]. LLV further tries to improve its effectiveness in using the CSP approach by considering, what Keng refers to as the *cruciality* of the solution, that measures how a solution impacts the tasks not selected and what he terms as the *least impact policy* [30]. Therefore, the scheduling metric for LLV considers both the value gained, (PGV) and the value lost, (PLV), to produce a scheduling metric *Net Lost Value*.

LLV will order jobs with the lowest *Net Lost Value*, first, to produce a schedule that is aimed at extracting increased value for the user, by limiting the net value lost (for the jobs not selected) while increasing the net value gained (for the selected job).

Formally, given a set of jobs J, the *Net Lost Value* (NLV) for selecting job j_i is defined as follows:

$$NLV(j_i) = PLV(j_i) - PGV(j_i) \tag{1}$$

As this scheduling policy will need to work in a big data environment where the task duration cannot be predicted in any amount of certainty, both PLV and PGV will use the Expected Value from the task duration distribution's probability density function and the TVD function, in its calculation. Therefore, the Expected Value (EV) for job j_i with TVD Function V_i, current time t_c and a probability density function $f_i(t)$ of the task duration distribution, is defined as follows:

$$EV(j_i, t_c) = \int_{t_c}^{\infty} V_i(t).f_i(t - t_c)\, dt \tag{2}$$

The *Potential Lost Value*(PLV) of selecting job j_i is the sum of the lost value of all other remaining jobs (j_k) not selected for processing, defined as:

$$PLV(j_i) = \sum_{j_k \in J, k \neq i, k=1}^{n} \big(EV(j_k, t_c) - EV(j_k, t_c + p_i)\big) \tag{3}$$

where:
n: number of jobs in set of job J
t_c: current time
p_i: non-weighted expected value of the task duration distribution for job j_i given probability density function $f_i(t - t_c)$, defined as:

$$p_i = \int_{t_c}^{\infty} (t - t_c).f_i(t - t_c)\, dt \tag{4}$$

The *Potential Gain Value* (PGV) of selecting job j_i is the difference of value between processing job j_i at current time t_c versus at later time, $t_c + \overline{p_i}$, defined as

$$PGV(j_i) = (EV(j_i, t_c) - EV(j_i, t_c + \overline{p_i})) \tag{5}$$

where $\overline{p_i}$ is calculated using the non-weighted expected value of the task duration distribution of the jobs not selected k with probability density function $f_k(t - t_c)$, defined as:

$$\overline{p_i} = (\sum_{k \neq i, k=1}^{n} p_k)/(n-1) \tag{6}$$

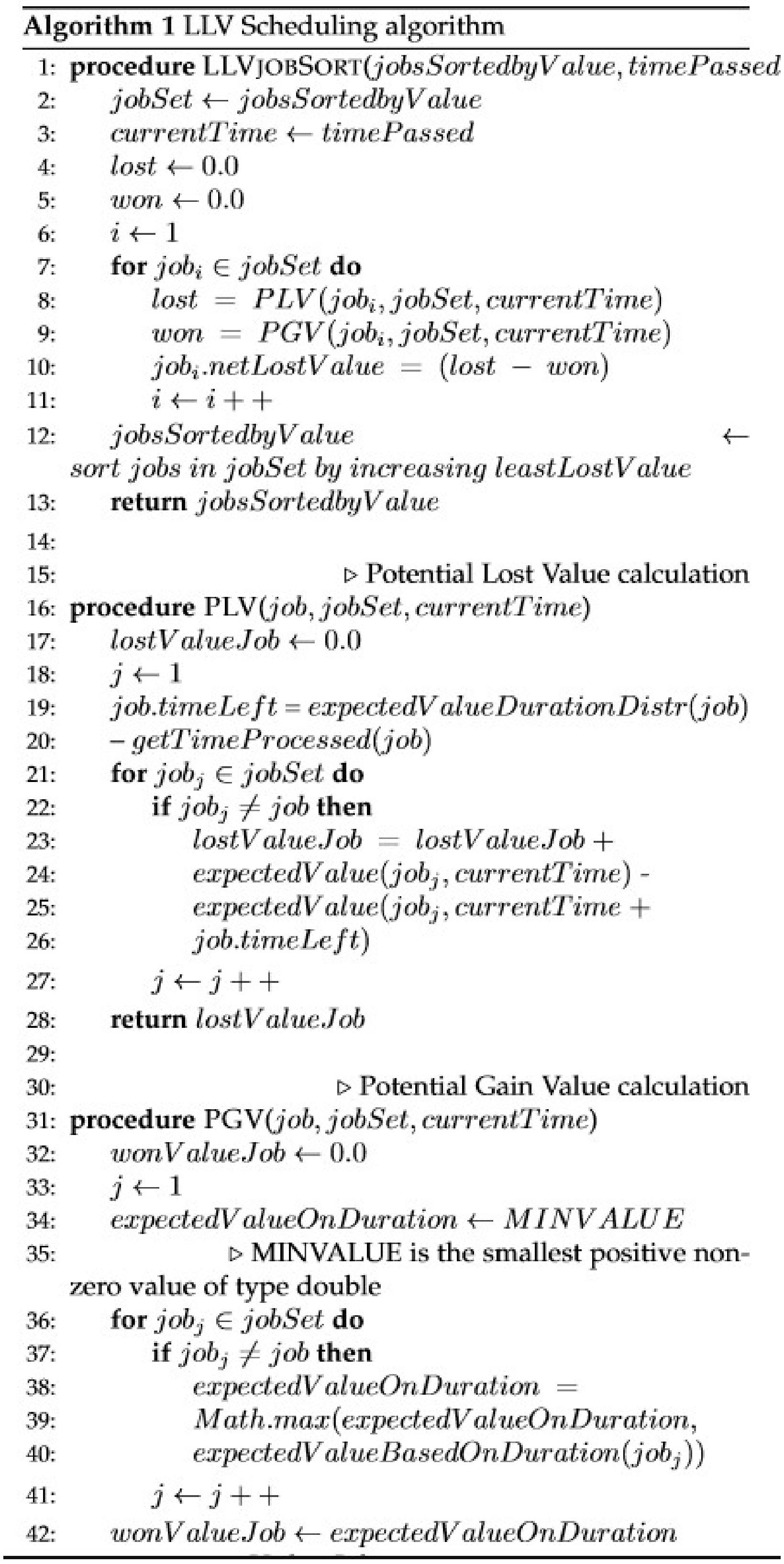
and

$$p_k = \int_{t_c}^{\infty} (t - t_c).f_k(t - t_c)\, dt \tag{7}$$

One advantage of PLV is that in considering the value lost from the jobs not selected it inadvertently considers discontinuity of value in the TVD function, described in Figure 2 as

well as impending penalties, where TVD function starts to accrue negative value. Therefore, jobs within the workload are likely to be scheduled for completion, before the critical value loss discontinuities and penalties.

Algorithm Least Lost Value (LLV)



Algorithm 1 LLV Scheduling algorithm

```
procedure LLVjobSort($jobsSortedbyValue, timePassed$
    $jobSet \leftarrow jobsSortedbyValue$
    $currentTime \leftarrow timePassed$
    $lost \leftarrow 0.0$
    $won \leftarrow 0.0$
    $i \leftarrow 1$
    for $job_i \in jobSet$ do
        $lost = PLV(job_i, jobSet, currentTime)$
        $won = PGV(job_i, jobSet, currentTime)$
        $job_i.netLostValue = (lost - won)$
        $i \leftarrow i++$
    $jobsSortedbyValue \leftarrow$
sort jobs in jobSet by increasing leastLostValue
    return $jobsSortedbyValue$

                              ▷ Potential Lost Value calculation
procedure PLV($job, jobSet, currentTime$)
    $lostValueJob \leftarrow 0.0$
    $j \leftarrow 1$
    $job.timeLeft = expectedValueDurationDistr(job)$
    $- getTimeProcessed(job)$
    for $job_j \in jobSet$ do
        if $job_j \neq job$ then
            $lostValueJob = lostValueJob +$
            $expectedValue(job_j, currentTime)$ -
            $expectedValue(job_j, currentTime +$
            $job.timeLeft)$
        $j \leftarrow j++$
    return $lostValueJob$

                              ▷ Potential Gain Value calculation
procedure PGV($job, jobSet, currentTime$)
    $wonValueJob \leftarrow 0.0$
    $j \leftarrow 1$
    $expectedValueOnDuration \leftarrow MINVALUE$
                    ▷ MINVALUE is the smallest positive non-
zero value of type double
    for $job_j \in jobSet$ do
        if $job_j \neq job$ then
            $expectedValueOnDuration =$
            $Math.max(expectedValueOnDuration,$
            $expectedValueBasedOnDuration(job_j))$
        $j \leftarrow j++$
    $wonValueJob \leftarrow expectedValueOnDuration$
```

Author: Shireen Seakhoa-King *17/06/2018*

Code Extract: START

```
	//LLVRa - actual final LLVRa
		public static final TimeValueComparator LeastLostValueAccRA = new TimeValueComparator("LLVRA") {
			private double lostValue(Job job, Collection<Job> allJobs, long time) {
				double lostValue = 0;
				long timeLeft = (long)expectedValueBasedOnDuration(job) - job.getTimeProcessed();
				for(Job j : allJobs) {
					if(!j.equals(job)) {
						lostValue += expectedValue(j, time)
								- expectedValue(j,  time + timeLeft);
					}
				}

				double expectedValueOnDuration = Double.MIN_VALUE;
				for(Job j : allJobs){
					if(!j.equals(job)) {
						expectedValueOnDuration = Math.max(expectedValueOnDuration,
expectedValueBasedOnDuration(j));
					}
				}

				double wonValue = expectedValue(job, time)
								 - expectedValue(job, time + (long)expectedValueOnDuration);

				return lostValue - wonValue;
			}
			@Override
			public List<Job> sort(List<Job> jobs, long time, int machinesAvailable) {

				Set<Job> jobsSet  = new HashSet<>(jobs);
				Comparator<Job> comp = (j1, j2) -> Double.compare(lostValue(j1, jobsSet, time), lostValue(j2,
jobsSet, time));
				jobs.sort(comp);

				return jobs;
			}
		};
	public static double expectedValue(Job j, long time) {
		AbstractRealDistribution duration = j.getDurationDistribution();

		double lowerBound = duration.getSupportLowerBound();
		double upperBound = duration.getSupportUpperBound();
		double mean = duration.getNumericalMean();

		// MULTI TVDFUNCTION CALCULATION
		// SSK - EV() =  Integ(f(t-tc)*vi(t))  lowbound - upperbound
		//
		double wholeMedianValue = 0;
		for(int num=0; num<j.getMultiTvdFunc().size(); num++){
			double medianValue = 0;
			double i;
			for(i = mean; i < upperBound; i++) {
				double prob = duration.density(i);

				// SSK - f(t-tc)*vi(t)
				medianValue += prob * j.getSingleValue((long) (time - j.getTimeProcessed() + i), 
j.getMultiTvdFunc().get(num), j.getWeightFuc().get(num));
				if(prob < 0.00001) {
					break;
				}
			}

			for(i = mean-1; i > lowerBound; i--) {
				double prob = duration.density(i);
				medianValue += prob * j.getSingleValue((long) (time - j.getTimeProcessed() + i), 
j.getMultiTvdFunc().get(num), j.getWeightFuc().get(num));
				if(prob < 0.00001) {
					break;
				}
			}
			wholeMedianValue += medianValue;
		}

		return wholeMedianValue;
	}
```

```
public static double expectedValueBasedOnDuration(Job j) {
        AbstractRealDistribution duration = j.getDurationDistribution();

        double lowerBound = duration.getSupportLowerBound();
        double upperBound = duration.getSupportUpperBound();
        double mean = duration.getNumericalMean();


                double medianValue = 0;
                double i;

                 for(i = mean; i < upperBound; i++) {
                        // f(t-tc)
                        double prob = duration.density(i);

                        // f(t-tc)*(t-tc)
                        medianValue += prob * i;
                        if(prob < 0.00001) {
                                break;
                        }
                }

                for(i = mean-1; i > lowerBound; i--) {
                        double prob = duration.density(i);
                        medianValue += prob * i;
                        if(prob < 0.00001) {
                                break;
                        }
                }

                return medianValue;
}
```

Code Extract: END