Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior

Ville Satopää[†], Jeannie Albrecht[†], David Irwin[‡], and Barath Raghavan[§]
[†]Williams College, Williamstown, MA
[‡]University of Massachusetts Amherst, Amherst, MA
[§]International Computer Science Institute, Berkeley, CA

***Abstract*—Computer systems often reach a point at which the relative cost to increase some tunable parameter is no longer worth the corresponding performance benefit. These "knees" typically represent beneficial points that system designers have long selected to best balance inherent trade-offs. While prior work largely uses ad hoc, system-specific approaches to detect knees, we present Kneedle, a general approach to online and offline knee detection that is applicable to a wide range of systems. We define a knee formally for continuous functions using the mathematical concept of curvature and compare our definition against alternatives. We then evaluate Kneedle's accuracy against existing algorithms on both synthetic and real data sets, and evaluate its performance in two different applications.**

I. Introduction

Selecting the "right" operating point for a given system is often thought of as an art form, since the direct and indirect costs and benefits of changing different system parameters are difficult or even impossible to quantify. For example, an important operating point in a large MapReduce job occurs when the job should no longer wait for "slow" tasks to finish, but instead speculatively re-execute work on other nodes in hopes of finishing the job sooner [1]. Since MapReduce's goal is to finish all tasks as fast as possible, it must decide when the cost, in terms of a job's running time and cluster utilization, is worth the corresponding performance benefit, in terms of task completion percentage. Congestion-responsive network protocols face a related challenge when setting a sending rate: a protocol must decide a rate that maximizes performance without exceeding its fair share and causing congestion.

In prior work, the issue has frequently been couched as identifying one or more "knees"—operating points, based on recent trends, where the perceived cost to alter a system parameter is no longer worth the expected performance benefit. For MapReduce, triggering speculative execution after observing a knee in the task completion percentage ensures that the system re-executes tasks that are significantly slower than other similar tasks that have finished execution. In the case of a network protocol, successive increases to the sending rate should cease if delay signals congestion by increasing steeply, forming a knee. However, while the problem of knee detection—finding "good" operating points in system behavior—seems straightforward, to the best of our knowledge there exists neither an accepted definition of a knee nor a general systematic approach for detecting one.

Numerous researchers in widely disparate areas frequently encounter knee detection problems similar to those we describe [1], [2], [3], [4], [5]. In these systems, researchers either use ad hoc or system-specific approaches to detect knees, or defer the problem to future work. While a finely-crafted system-specific approach will perform better than a general knee detection approach, a designer may not take the time to design one. Thus, our aim is not to improve or optimize a specific system or protocol, but to provide system designers a general tool for improving the parts of their system they generally do not take the time to optimize. In network protocol and system design, rules-of-thumb often serve researchers and operators well in the absence of an optimal solution. We believe that a tool for knee detection adds to their problem solving arsenal. Our hypothesis is that a knee detection algorithm that does not require tuning for a specific system or operational characteristics is applicable in a wide range of settings where developers do not take the time to design, test, and optimize a system-specific algorithm.

II. Defining and Detecting Knees

While the notion of a knee is well-known, we are not aware of a broadly accepted definition in prior literature. The confusion stems from the fact that researchers, in many cases unknowingly, use knees as a substitite for a more comprehensive cost-benefit analysis that is either difficult or impossible to perform. Performing a direct cost-benefit analysis is often complex, since it is inherently system-, platform-, and workload-specific. Further, many systems are not predictable due to volatile operating conditions.

For example, unpredictable failure rates in large clusters, which may change over time, are the root cause of stragglers in MapReduce jobs [1]. Likewise, since multiple flows share network links in the Internet, network protocols cannot predict in advance the rapidly changing level of TCP-friendly bandwidth available, but must instead continuously adapt to the indirect signals of packet loss and delay [6]. In lieu of a complex system-specific analysis, operators tend to select operating points, or knees, that are "good enough" by observing where performance improvements start to level off as a function of one or more tunable system parameters. Note that we focus on knee detection for complex systems that change their behavior according to volatile, and potentially unpredictable, operating conditions, and not for simple systems that permit standard closed-form models, e.g., M/M/1 queues [7].

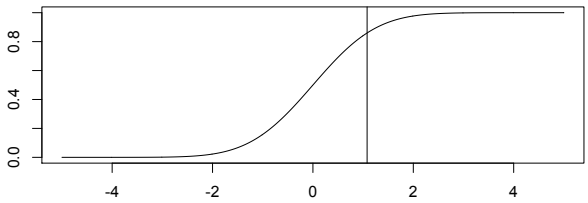


Fig. 1: CDF of a standard Gaussian distribution with mean=0 and standard deviation=1. Vertical bar indicates point of maximum curvature. The inflection point of this curve occurs at $x = 0$.

A. Knee Definition

The difficulty with defining a knee formally is that "good enough" in one system may not be "good enough" in another. Since knees only serve as an approximation, operators interpret them differently in different situations. Thus, knee detection is an inherently heuristic process. However, to design a general application-independent knee detection algorithm, we require a consistent definition applicable to any system. In this work, as in [8], we use the mathematical definition of curvature for a continuous function as the basis for our knee definition. For any continuous function f, there exists a standard closed-form $K_f(x)$ that defines the curvature of f at any point as a function of its first and second derivative:

$$K_f(x) = \frac{f''(x)}{(1 + f'(x)^2)^{1.5}}$$

The point of maximum curvature is well-matched to the ad hoc methods operators use to select a knee, since curvature is a mathematical measure of how much a function differs from a straight line. As a result, maximum curvature captures the leveling off effect operators use to identify knees. Importantly, unlike other common definitions, curvature is application-independent and (i) does not depend on the relationship between system parameters and performance, or (ii) require setting system-specific thresholds. Note that knee detection does depend on the selection of proper adjustable system parameters and performance metrics, as we show for our examples in Section V.

It is important to realize why a knee definition based only on the first derivative is not enough to identify a knee. Consider the simple example in Figure 1, where the y-axis represents some performance metric, the x-axis represents a tunable system parameter, and the vertical bar represents the point of maximum curvature. The maximum of the first derivative is the inflection point of the curve, which occurs at $x = 0$ in Figure 1. The inflection point is not representative of the knee since performance continues to improve significantly beyond it. Instead, the inflection point only captures where the rate of performance increase reaches a maximum. In contrast, the curvature definition precisely matches the concept of a knee. [8] includes a survey of a range of other knee definitions from prior work, primarily in the context of clustering algorithms [7], [9], [10], [11], [12]. We discuss alternative definitions below.

While curvature is well-defined for continuous functions, it is not well-defined for discrete data sets. In the discrete case, we could determine curvature by fitting a continuous function to the data and using the function's point of maximum curvature. However, fitting a continuous function to a set of arbitrary data points is difficult, especially if the data is noisy. Further, determining the maximum curvature of the resulting function may not be sufficient, since the curvature at any point of a function is dependent on the entire function, including points not in the relevant data set. Thus, maximum curvature may fall outside the data's valid range or be one of the set's end-points. Since an approximation of curvature requires at least three points—the minimum number of points that define a circle—end-points in a data set do not have curvature values by definition. Thus, using the closed-form formulation as a direct basis for knee detection on discrete data is not possible.

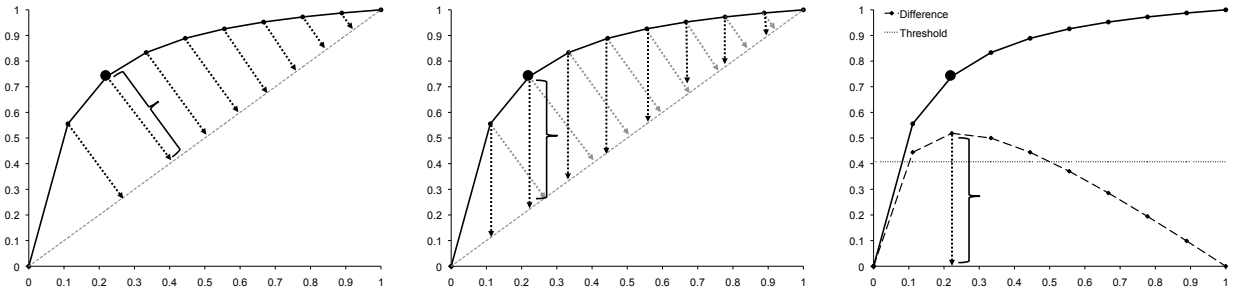
B. Knee Detection in Discrete Data Sets

Researchers have proposed multiple previous approaches to detecting knees in discrete data. Before formulating our curvature-inspired algorithm in Section III, we present two existing approaches—Angle-based and EWMA—from prior research for comparison, as well as another approach we formulate based on Menger curvature, a direct discrete equivalent of continuous curvature. Note that the Angle-based and Menger algorithms are designed specifically for offline cases, where the entire data set is known in advance, while EWMA is designed to detect knees online as data points become known.

Angle-based. The geometric "angle-based" approach of Zhao *et al.* [13] is an extension of the L-method for detecting knees in clustering applications [8]. The Angle-based approach first finds the local minima of the successive differences $(y_1 + y_3 - 2y_2)$ for each consecutive triple of points. For example, consider a straight line that goes through the consecutive points (x_1, y_1), (x_2, y_2), and (x_3, y_3). Assuming x-values are evenly spaced, then $y_1 + y_3 - 2y_2 = 0$ for any straight segment. However, if these three points form a knee, (x_2, y_2) must be above the the straight line that goes through (x_1, y_1) and (x_3, y_3). In this case $y_1 + y_3 - 2y_2 < 0$. "Sharper" knees have more negative difference values.

Next, since successive differences are local measures and ignore the overall trend of the curve, the algorithm combines the differences with an *angle value*. After obtaining the local minima of the successive differences, the algorithm sorts the minima, and, starting from the point with the largest difference value, calculates the two angles formed by the y-axis and the line going through each successive pair of points associated with the corresponding difference value. The sum of these two angles is the angle value. Knees are detected at the local maxima of these angle values.

Menger Curvature. While curvature is not well-defined for arbitrary discrete data sets, Menger curvature defines the curvature for three discrete points as the curvature of the circle circumscribed about those points [14]. Thus, we define the Menger curvature for each point $p_i = (x_i, y_i)$ in an n point data set as being equal to $1/r$ for the circle of radius r circumscribed about p_1, p_i, and p_n. The curvature of the circumscribed circle is straightforward to compute and



(a) (b) (c)

Fig. 2: Kneedle algorithm for online knee detection. (a) depicts the smoothed and normalized data, with dashed bars indicating the perpendicular distance from $y = x$ with the maximum distance indicated. (b) shows the same data, but this time the dashed bars are rotated 45 degrees. The magnitude of these bars correspond to the difference values used in Kneedle. (c) shows the plot of these difference values and the corresponding threshold values (with $S = 1$). The knee is found at $x = 0.22$ and is detected after receiving the point $x = 0.55$.

is simply a function of the lengths of the sides of the triangle with the points as vertices. However, as we show in Section IV, while Menger closely approximates curvature for offline data drawn from ideal continuous functions, it does not work well for the noisy online data sets typical of computing systems.

EWMA. The EWMA approach uses techniques similar to those employed by Bollinger Bands [15] and Geometric Moving Average algorithms for change detection [16]. The algorithm that we use is based on the methodology described by Albrecht *et al.* in their work on partial barriers [3], which derives from previous work on MONET [17]. EWMA is an online algorithm that uses two exponentially weighted moving averages. The first EWMA, called arr, is used to smooth the input data, which is viewed as host arrival times. The second EWMA, $arrvar$, keeps track of the average deviation from arr, and is an estimate of the variance in arrival times. Finally, these two values are used to compute a maximum wait threshold of $arr + 4 \cdot arrvar$, which represents the maximum amount of time to wait for the next point to arrive. If the point arrives after this threshold, or the threshold is reached without seeing the next arrival, EWMA declares a knee. One important attribute of this algorithm is that EWMA does not directly report where the knee point is—it only determines if a knee has been passed. As a result, EWMA is only applicable in an online setting.

III. Kneedle Algorithm

Kneedle is based on the notion that the points of maximum curvature in a data set—the knees—are approximately the set of points in a curve that are local maxima if the curve is rotated θ degrees clockwise about $(x_{\mathsf{min}}, y_{\mathsf{min}})$ through the line formed by the points $(x_{\mathsf{min}}, y_{\mathsf{min}})$ and $(x_{\mathsf{max}}, y_{\mathsf{max}})$. We choose this line because we want to preserve the overall behavior of the data set—using a line of best fit, for example, risks cutting off the end points due to a higher concentration of points in the middle of the curve. After rotating about this line, the local maxima—and thus knees—are the points at which the curve differs most from the straight line segment connecting the first and last data point, thereby approximating the point of maximum curvature for a discrete set of points. Since maximum curvature is an inherent measure of the point where a continuous function differs most from a straight line, Kneedle uses a literal measure of the point that differs most from the straight line connecting the set's end-points.

Figure 2 depicts how Kneedle works for data points drawn from the curve $y = -1/x + 5$ where x-values are between 0 and 1. Note that we assume that the curves under consideration have negative concavity. For curves with consistently positive concavity (*e.g.*, forming "elbows" rather than knees) it is trivial to invert the graph by replacing each y_i with $y_{\mathsf{max}} - y_i$ and x_i with $x_{\mathsf{max}} - x_i$.

We summarize Kneedle below. Put simply, knees occur when a curve becomes more "flat," indicating a decrease in curvature. The algorithm works as follows:

1. First we use a smoothing spline to preserve the shape of the original data set as much as possible, although other smoothing techniques, such as an exponentially weighted moving average, could also be used. Let D_s represent the finite set of x- and y-values that define a smooth curve, *i.e.,* one that has been fit to a smoothing spline.

$$D_s \quad = \quad \{(x_{s_i}, y_{s_i}) \in \mathbb{R}^2 \mid x_{s_i}, y_{s_i} \geq 0\}.$$

2. We want our algorithm to function in the same way regardless of the magnitude of the values in the underlying data. Thus, we next normalize the points of the smooth curve to the unit square, as shown in Figure 2(a). This does not change the shape or trends of the data set:

$$\begin{aligned} D_{\mathsf{sn}} &= \{(x_{\mathsf{sn}_i}, y_{\mathsf{sn}_i})\}, \text{ where} \\ x_{\mathsf{sn}_i} &= (x_{s_i} - \min\{x_s\})/(\max\{x_s\} - \min\{x_s\}), \\ y_{\mathsf{sn}_i} &= (y_{s_i} - \min\{y_s\})/(\max\{y_s\} - \min\{y_s\})\}. \end{aligned}$$

3. Next, we let D_d represent the set of differences between the x- and y-values, *i.e.,* the set of points $(x,\ y - x)$ as illustrated in Figure 2(b). The goal is to find out when the difference curve changes from horizontal to sharply decreasing, since this indicates the presence of a knee in the original data set. Note that the actual values of the difference points are irrelevant. We are only interested in observing the trends of the difference curve, as seen in Figure 2(c).

$$\begin{aligned} D_d &= \{(x_{d_i}, y_{d_i})\}, \text{ where} \\ x_{d_i} &= x_{\mathsf{sn}_i}, \\ y_{d_i} &= y_{\mathsf{sn}_i} - x_{\mathsf{sn}_i}. \end{aligned}$$

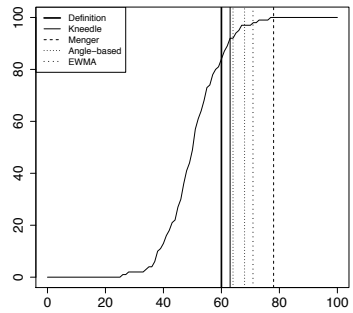


Fig. 3: Kneedle, Menger, Angle-based, and EWMA for synthetic data set. Maximum curvature occurs at $x = 60$.

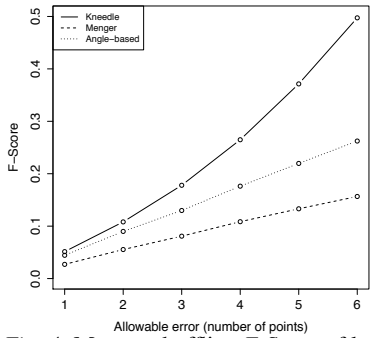


Fig. 4: Measured offline *F-Score* of knee detection algorithms using NoisyGaussian data.

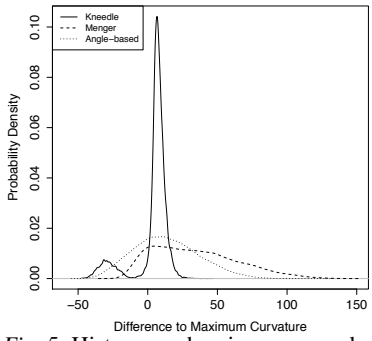


Fig. 5: Histogram showing measured offline distances (numbers of x-values) to "correct" knees.

4. To find the knee points in the normalized curve, *e.g.*, the places where the curve flattens out, we calculate the local maxima of the difference curve. These points indicate the instances where the rate of increase of y begins to decrease. Each of these local maximum points are a candidate knee point in the original data curve:

$$\begin{aligned} D_{\mathsf{lmx}} &= \{(x_{\mathsf{lmx}_i}, y_{\mathsf{lmx}_i})\}, \text{ where} \\ x_{\mathsf{lmx}_i} &= x_{d_i}, \\ y_{\mathsf{lmx}_i} &= y_{d_i} \mid y_{d_{i-1}} < y_{d_i},\ y_{d_{i+1}} < y_{d_i}. \end{aligned}$$

5. For each local maximum $(x_{\mathsf{lmx}_i}, y_{\mathsf{lmx}_i})$ in the difference curve, we define a unique threshold value, T_{lmx_i}, that is based on the average difference between consecutive x-values and a sensitivity parameter, S. The sensitivity parameter allows us to adjust how aggressive we want Kneedle to be when detecting knees. Smaller values for S detect knees quicker, while larger values are more conservative. Put simply, S is a measure of how many "flat" points we expect to see in the unmodified data curve before declaring a knee. We explore the choice of S in Section IV. In Figure 2(c), the threshold line is plotted with $S = 1$.

$$T_{\mathsf{lmx}_i} = y_{\mathsf{lmx}_i} - S \cdot \frac{\sum_{i=1}^{n-1} \left(x_{\mathsf{sn}_{i+1}} - x_{\mathsf{sn}_i}\right)}{n-1}$$

6. If any difference value (x_{d_j}, y_{d_j}), where $j > i$, drops below the threshold $y = T_{\mathsf{lmx}_i}$ for $(x_{\mathsf{lmx}_i}, y_{\mathsf{lmx}_i})$ before the next local maximum in the difference curve is reached, Kneedle declares a knee at the x-value of the corresponding local maximum $x = x_{\mathsf{lmx}_i}$. If the difference values reach a local minimum and starts to increase before $y = T_{\mathsf{lmx}_i}$ is reached, we reset the threshold value to 0 and wait for another local maximum to be reached.

Note that Kneedle can be run offline or online. In the online case, Kneedle can "correct" old knee values if necessary as points are received. Kneedle's online run time for any given n pairs of x- and y-values is bounded by $\sum_{i=1}^{n} i = O(n^2)$.

IV. Evaluating Kneedle

We compare the performance of Kneedle to the offline (Angle-based, Menger) and online (EWMA) algorithms separately, since their goals are different. In offline settings, our aim is to determine a base-line accuracy for each algorithm using synthetic data sets drawn from continuous functions where the true knees are well-known. After showing that Kneedle closely approximates the true knees, we then compare its online behavior against EWMA to evaluate how quickly it is able to detect knees once they "appear" in the data.

A. Detecting Knees in Synthetic Data Sets

To evaluate Kneedle, we developed a synthetic data source which we call NoisyGaussian that yields data similar to many of the real data sets of interest, but allows us to vary the overall shape of the curve. To generate a NoisyGaussian, we start with a Gaussian function with a randomly selected standard deviation and mean. Then we generate the NoisyGaussian data set using the cumulative count of the randomly generated points whose value is less than x. The resulting curve is similar to a Gaussian cumulative distribution function in overall shape.

The benefit of evaluating the knee detection algorithms using NoisyGaussian is that an approximate closed-form solution exists for the point of maximum curvature. We derive the point of maximum curvature by computing it for the underlying Gaussian CDF in terms of standard deviation σ and mean μ. Although we omit the details for brevity, the point of maximum curvature is approximately $x \approx \mu + \sigma$ with a small bounded error. We use this closed-form expression to represent the "correct" knee in our evaluation.

To illustrate the general behavior of each knee detector, we plot the knees each algorithm detects in Figure 3 for a sample NoisyGaussian data set with $\mu = 50$ and $\sigma = 10$.

B. Offline Accuracy

To evaluate offline accuracy, we use three common statistical metrics: precision, recall, and F-Score. Precision measures the correctness of each knee an algorithm detects. A low precision value indicates the presence of numerous false positives, where a false positive is any detected knee that does not align with maximum curvature. Recall measures completeness by quantifying the percentage of correct knees an algorithm detects out of the total number of correct knees. Note, however, that recall does not penalize for incorrect detections. Our third metric, F-Score, is the harmonic mean of precision and recall. Since an ideal knee detection algorithm has both high recall

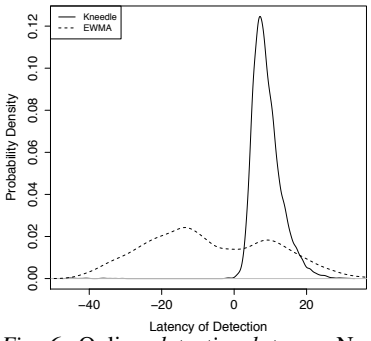


Fig. 6: Online *detection latency*. Negative values indicate early detections.

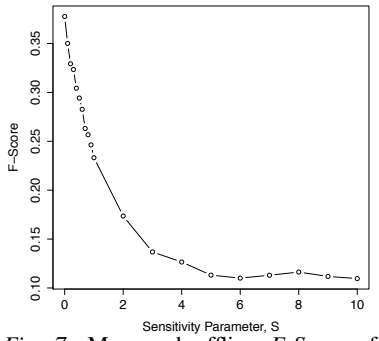


Fig. 7: Measured offline *F-Scores* for varying sensitivity values in Kneedle.

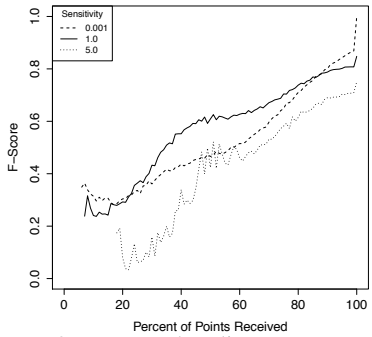


Fig. 8: Measured online *F-Scores* for varying sensitivity values in Kneedle.

and high precision, we use F-Score to capture both measures of accuracy in a single value. An F-Score value of 1 is best.

To evaluate our algorithms, we generate 10,000 NoisyGaussian data sets. Since none of the algorithms detect knees at exactly the point of maximum curvature, we vary how many data points we allow for error. For example, suppose our data set includes points at $x = 1, 2, 3, 4, 5$, and the point of maximum curvature is $x = 4$. With an allowable error of 1, we declare the algorithm as finding a "correct" knee if it detects a knee at $x = 3$, 4, or 5. Figure 4 shows that Kneedle's F-Score is better than the Angle-based or Menger algorithm.

Using the closed-form approximation for the point of maximum curvature in our NoisyGaussian data sets, we can identify "true" knees in the data. This allows us to quantify the accuracy of each algorithm by measuring the distance, in terms of the number of x-values, between the true knees and the detected knees. Figure 5 shows the results of measuring the distance, in terms of the number of x-values, between the true knees and the detected knees. In this histogram, we see that Kneedle approximates the point of maximum curvature much more closely than either Menger or Angle-based, since the density of the histogram is highest between 0 and 25, while Menger and Angle-based show a wider variation.

C. Online Detection Latency

In this section, we evaluate detection latency—the number of data points beyond the knee required for detection—for both EWMA and Kneedle. For online Kneedle, we execute the knee detection algorithm after receiving each new data point, in order of increasing x. For both EWMA and Kneedle, we compute the detection latency as the number of data points between when the algorithm detects a knee and the actual knee point as determined by the point of maximum curvature. For example, suppose the data set has points at $x = 1, 2, 3, 4$, and 5, with a true knee at $x = 3$. Now suppose that after receiving the point at $x = 5$, the knee detection algorithm detects a knee. In this case, we compute the the latency as $5 - 3 = 2$. In Figure 6 we plot a histogram of the detection latency for EWMA and Kneedle with $S = 1$. The experiment highlights the fact that Kneedle rarely has a significant detection latency, while EWMA often has high detection latencies.

D. Sensitivity

To better understand the importance of sensitivity, S, to Kneedle's performance, we again use F-Score. Figures 7 and 8 show the results of our sensitivity analysis in offline and online settings respectively. In both graphs, we compute Kneedle's F-Score using a wide range of sensitivity values. We compare the F-Score from 10,000 data sets for each value of S. In the offline graph, we use the points of maximum curvature as the true knees, and compute the F-Score based on those values. In the online graph, our goal is to determine how quickly Kneedle approaches the offline case, and thus we use the knees detected by offline Kneedle as the correct knees. Not surprisingly, in offline settings where Kneedle has perfect information, the highest F-Score occurs when $S = 0$. In online settings, the results vary depending on the number of points received, but overall $S = 1$ has the best results.

V. Application Results

This section demonstrates Kneedle's usefulness in real applications. First, we identify knees in a data set from prior work, and show that we find close to the same knees that the authors found with system-specific techniques. Next we evaluate Kneedle's performance for two sample applications: a MapReduce-like system and a TCP-friendly network protocol.

A. Using Kneedle in Existing Applications

Figure 9 applies knees to object replication, where the knees represent the optimal degrees of replication for high availability given various object distributions (data from Figure 5 in [5]). The application requires the detection of multiple knees in object popularity curves, each of which has considerable noise. Unlike other knee detection algorithms, such as Menger, Kneedle is capable of detecting multiple knees, where the sensitivity of this detection depends on the selected value of S. Note that we consider this knee detection application to be offline, since Zhong *et al.* observe: "[w]e expect that the replica adjustment overhead due to object request popularity changes would not be excessive in practice...our analysis of real system object request traces in Section 3.2 suggests that the popularities of most data objects tend to remain stable over multi-week periods." The knees found by Kneedle in this graph concur with those identified by the original authors.

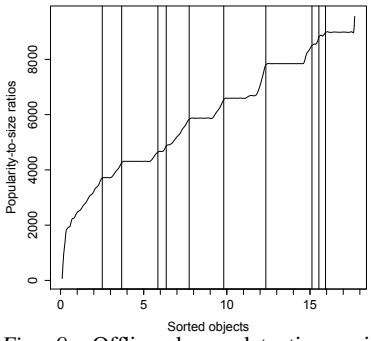


Fig. 9: Offline knee detection using Kneedle for replication [5].

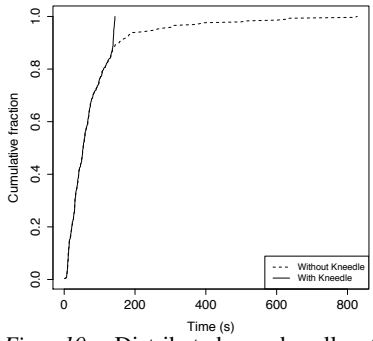


Fig. 10: Distributed work allocator with/without online Kneedle.

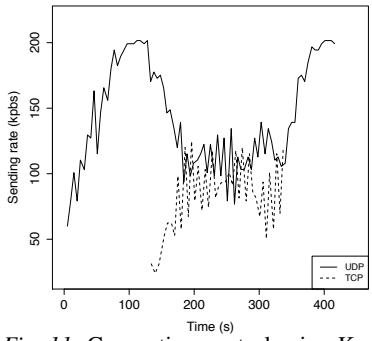


Fig. 11: Congestion control using Kneedle (UDP) vs. TCP.

B. Using Kneedle for Speculative Execution

To test the effectiveness of Kneedle in our own MapReduce-like setting, we integrated our algorithm into a prototypical distributed batch computing system that farms out tasks to PlanetLab nodes [18]. We ran an experiment in which we gave 300 PlanetLab nodes a small task that required a mix of CPU and I/O resources to process. We then used Kneedle to find knee points and reallocate tasks from slow nodes to fast nodes, reducing the overall task completion time. This task reallocation is similar to MapReduce's strategy for coping with the presence of stragglers. Figure 10 demonstrates that Kneedle can be successfully integrated into existing systems with minimal effort: the only change required to our work allocation system was a single function call. That is, each time a task completed, we called Kneedle with the new data point. When Kneedle returned a knee, we simply reallocated unfinished tasks to idle nodes, reducing the total completion time from 827 seconds down to 143 seconds.

C. Using Kneedle for Congestion Control

Next we show the applicability of Kneedle to an entirely different domain: network congestion control. Here we use Kneedle to find the knee, as described by Jain [2], in the offered load versus packet delay curve. We implemented a TCP-friendly congestion control algorithm for UDP flows, similar to the TFRC [6], but without the careful measurement of packet transmissions and complicated equations. For each reply packet from the receiver, our algorithm simply calculates the round-trip delay. We increment the rate every time a packet is transmitted and pace the packets evenly; for every 100 packets sent, we compute the knee point and use it as the new target rate. Figure 11 shows the behavior of our UDP sender versus an ordinary TCP flow that joins and then departs. We use dummynet to limit bandwidth to 200Kbps. We find that our simple algorithm is both able to stabilize to the bottleneck bandwidth and to share bandwidth fairly with a TCP flow. While this experiment is simplistic, it highlights the benefits of Kneedle's general approach.

VI. Conclusion

In this paper, we present a formal definition for a knee in discrete data sets based on the mathematical definition of curvature for continuous functions. We propose a new algorithm, Kneedle, for approximating these knee points in discrete data sets, and compare with existing knee detection approaches using both synthetic and real application data sets. We then show that Kneedle is useful in real systems by integrating it with minimal effort into two distinct systems that encounter the knee detection problem. We believe Kneedle addresses a common problem that arises in systems research and engineering. In addition, we believe that our methodology for evaluating knee detection approaches will allow any future knee detection designs to be compared in a straightforward manner.

Acknowledgment

This material is supported by NSF grant CNS-0845349.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, 2004.
[2] R. Jain, "Congestion Control in Computer Networks: Issues and Trends," *IEEE Network*, vol. 4, no. 3, 1990.
[3] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat, "Loose Synchronization for Large-Scale Networked Systems," in *USENIX*, 2006.
[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," in *SIGCOMM*, 2003.
[5] M. Zhong, K. Shen, and J. Seiferas, "Replication Degree Customization for High Availability," in *EuroSys*, 2008.
[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *SIGCOMM*, 2000.
[7] C. Millsap, "Performance Management: Myths and Facts," Oracle Corporation, Tech. Rep., 1999.
[8] S. Salvador and P. Chan, "Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms," in *ICTAI*, 2004.
[9] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris, "A Robust and Scalable Clustering Algorithm for Mixed Type Attributes in Large Database Environments," in *KDD*, 2001.
[10] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *KDD*, 1996.
[11] A. Foss and A. Zaiane, "A Parameterless Method for Efficiently Discovering Clusters of Arbitrary Shape in Large Datasets," in *ICDM*, 2002.
[12] C. Millsap and J. Holt, *Optimizing Oracle Performance*. O'Reilly Media, Inc., 2003.
[13] Q. Zhao, V. Hautamaki, and P. Fränti, "Knee Point Detection in BIC for Detecting the Number of Clusters," in *ACIVS*, 2008.
[14] X. Tolsa, "Principal Values for the Cauchy Integral and Rectifiability," in *American Mathematical Society*, 2000.
[15] J. A. Bollinger, *Bollinger on Bollinger Bands*. McGraw-Hill, 2001.
[16] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Englewood Cliffs, N.J.: Prentice-Hall, 1993.
[17] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. N. Rao, "Improving Web Availability for Clients with MONET," in *NSDI*, 2005.
[18] L. L. Peterson, A. C. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in *OSDI*, 2006.