Reference

[1] D. Peteiro-Barral and B. Guijarro-Berdiñas (2012), “A survey of methods for distributed machine learning,” *Prog. Artif. Intell.*, vol. 2, no. 1, pp. 1–11.

[2] Jerry W. Thomas (2007), “Market Segmentation.”

[3] E. Bosco (2013), “Smorgasbord Of Segments,” *MediaPost Communications*. [Online]. Available: http://www.mediapost.com/publications/article/189477/a-smorgasbord-of-segments.html#axzz2HgNiNXbd. [Accessed: 23-Mar-2014].

[4] C. Moretti, K. Steinhaeuser, D. Thain, and N. V. Chawla (2008), “Scaling up Classifiers to Cloud Computers,” *2008 Eighth IEEE Int. Conf. Data Min.*, pp. 472–481.

[5] F. Provost and V. Kolluri (1999), “A survey of methods for scaling up inductive algorithms,” *Data Min. Knowl. Discov.*, vol. 42, pp. 1–42.

[6] A. J. C. Sharkey (1998), “Combining Arti cial Neural Nets Ensemble and Modular Multi-Net Systems.”

[7] D. Kriesel (2007), “A brief introduction to neural networks,” *Retrieved August*.

[8] B. Karlik and A. Olgac (2011), “Performance analysis of various activation functions in generalized MLP architectures of neural networks,” *… J. Artif. Intell. Expert …*, no. 1, pp. 111–122.

[9] K. Kumar and G. S. M. Thakur (2012), “Advanced Applications of Neural Networks and Artificial Intelligence: A Review,” *Int. J. Inf. Technol. Comput. Sci.*, vol. 4, no. 6, pp. 57–68.

[10] B. Kamgar-parsi, J. A. Gualtieri, J. E. Devaney, and B. Kamgar-parsi (1990), “Biological Cybernetics Clustering with Neural Networks,” vol. 208, pp. 201–208.

[11] D. S. Boone and M. Roehm (2002), “Retail segmentation using artificial neural networks,” *Int. J. Res. Mark.*, vol. 19, no. 3, pp. 287–301.

[12] A. SHARKEY (1996), “On combining artificial neural nets,” *Conn. Sci.*

[13] E. Durfee and J. Rosenschein (1994), “Distributed problem solving and multi-agent systems: Comparisons and examples,” *Ann Arbor*, pp. 1–10.

[14] K. S. Decker (1987), “Distributed problem-solving techniques: A survey,” *IEEE Trans. Syst. Man. Cybern.*, vol. 17, no. 5, pp. 729–740.

[15] H. S. Nwana (2009), “Software agents: an overview,” *Knowl. Eng. Rev.*, vol. 11, no. 03, p. 205.

[16] L. Panait and S. Luke (2005), “Cooperative Multi-Agent Learning: The State of the Art,” *Auton. Agent. Multi. Agent. Syst.*, vol. 11, no. 3, pp. 387–434.

[17] P. Stone and M. Veloso (2000), “Multiagent systems: A survey from a machine learning perspective,” *Auton. Robots*, pp. 1–57.

[18] H. Kitano and J. Hendler (1994), *Massively parallel artificial intelligence*. pp. 557–562.

[19] A. H. Bond (1988), “A Survey of Distributed Artificial Intelligence.”

[20] R. Prouty, S. Otto, and J. Walpole (1994), “Adaptive Execution of Data Parallel Computations on Networks of Heterogeneous Workstations Networks of Heterogeneous Workstations,” no. March.

Appendix A

SPADE Agent Development Environment

A.1 Simple Agent

```
import spade

class MessageSpaceAgent(spade.Agent.Agent):
      def _setup(self):
         print "MessageSpaceAgent starting . . ."

      def takeDown(self):
         pass
```

A.2Agent Behaviour

```
import spade

class MessageSpaceAgent(spade.Agent.Agent):
    class MyBehav(spade.Behaviour.Behaviour):
        def onStart(self):
            print "Starting behaviour . . ."

        def _process(self):
            pass

    def _setup(self):
        print "MessageSpaceAgent starting . . ."
        b = self.MyBehav()
        self.addBehaviour(b, None)

if __name__ == "__main__":
    a = MessageSpaceAgent("ms_agent@myhost.myprovider.com",
"secret")
    a.start()
```

A.3 ACL Messages

```
class MessageSpaceAgent (spade.Agent.Agent):
    class InformBehav(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            receiver =
spade.AID.aid(name="receiver@spade.domain.com")
            self.msg = spade.ACLMessage.ACLMessage()
            self.msg.setPerformative("inform") # Set the
"inform" FIPA performative
            self.msg.setOntology("myOntology") # Set the
ontology of the message content
            self.msg.setLanguage("JSON") # Set the language of
the message content
            self.msg.addReceiver(receiver)
            self.msg.setContent("msg content") # Set the
message content

            self.myAgent.send(self.msg)
```

Appendix B

Clustering Algorithm

B.1 Calculation of segment centroid

```
for p in 0 to K:
    Vsum = 0 #sum of the current segemnt
    for i in 0 to N:
        # Centroids
        Vsum += V[p][i]

    for j in 0 to dimn:
        y[p][j] = 0
        for i in 0 to N:
            y[p][j] += x[i][j] * V[p][i]

        y[p][j] /= Vsum

    # Residuals
    for i in 0 to N:
        R[p][i] = 0
        for j in 0 to dimn:
            tmp = x[i][j] - y[p][j]
            R[p][i] += tmp * tmp

for j in 0 to N:
    raVg = 0

    for i in 0 to M:
        raVg += R[i][j]
    raVg /= M

    C[j] = PC / raVg

y - segment centroids vector
dimn - no of segmentation attributes
x - input data vector
```

B.2 Parallel algorithm and the kernel launch

```
def dynamic(A, B, BIAS, c, v, r,unchange):

      i = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x

      if i >= N:
          return

      for p in 0 to K:
          _sum = 0.0
          for m in 0 to K:
              _sum +=  V[m][i]
            Api = - A * (_sum - V[p][i]) - B * (_sum - 1) -
                  C[i] * r[p][i] * V[p][i] - BIAS

          Vip = 1 / (1 + exp(-(Api - THETA) / RHO))
          offset = int(i*n) + j
          if math.fabs(vij - v[i, j]) < EPSILON:
                  unchange[offset] = 1
          V[p][i] = Vip

#kernal launch
thredPerBlock = (32, 1)
d_unchange = cuda.to_device(np.zeros(M * N)
dynamic[int((N + thredPerBlock[0] - 1) / thredPerBlock[0]),
thredPerBlock](A,B,BIAS,d_c,d_v,d_r,d_unchange)
```


B.3 Centroid Kernel Function

```
# vector multiplication of centroid calculation
def centroid_step_mult(j, k, x, v, c):
    i = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
    c[i] = x[i, j] * v[k, i]


#calculate residuals
def centroid_step_residuals(x,y,r):
    i = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x # no of
users
    k = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y # no
segments

    n = x.shape[0]
    m = y.shape[0]
    dimn = x.shape[1]

    if i >= n or k >= m:
        return

    r[k, i] = 0.0

    for j in range(0, dimn):
        tmp = x[i, j] - y[k, j]
        r[k, i] += tmp * tmp
```

B.4Host Centroid Kernel Launch Function

```
    def centroid(self):
        vsum = np.zeros(self.M)
        num_blocks = int(self.N/SUM_TBP) + (1 if (self.N % SUM_TBP)
else 0)
        self.y = self.d_y.copy_to_host()

        for i in range(0, self.M):
            vsum[i] = par_sum(self.d_v[i, :].reshape(self.N))

        for k in xrange(0, self.M): #desired segment vs customers
            for j in xrange(0, self.dimn):
                self.y[k, j] = 0
                d_c = cuda.to_device(np.zeros(self.N,
dtype=np.float32))
                centroid_step_mult[num_blocks, SUM_TBP](j, k,
self.d_x, self.d_v, d_c)
                _sum = par_sum(d_c)
                self.y[k, j] += _sum
                if vsum[k] > 0:
                    self.y[k, j] /= vsum[k]

        self.d_y = cuda.to_device(self.y)
        thredPerBlock = (SUM_TBP, 8)

        gridSize = (int((self.N + thredPerBlock[0] - 1) /
thredPerBlock[0]),
                    int((self.M + thredPerBlock[1] - 1) /
thredPerBlock[1]))

        centroid_step_residuals[gridSize, thredPerBlock](self.d_x,
self.d_y, self.d_r)
        thredPerBlock = SUM_TBP
        gridSize = int((self.N + thredPerBlock - 1) / thredPerBlock)
        centroid_step_update_c[gridSize,
thredPerBlock](np.float32(self.PC), self.d r, self.d c)
```

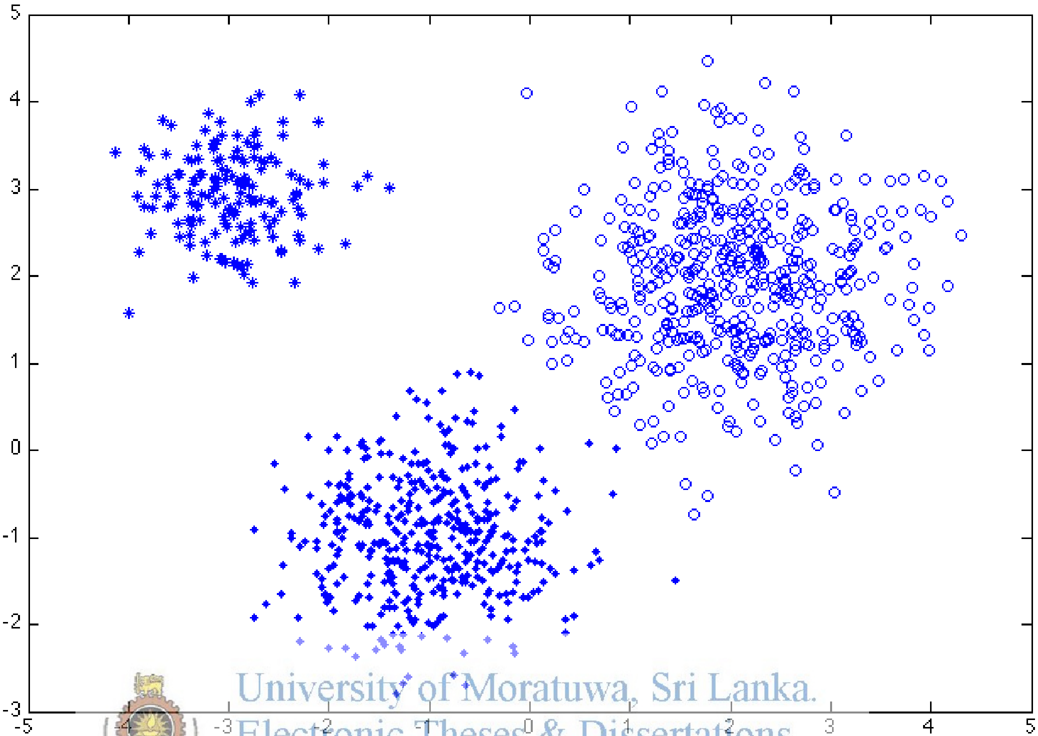
Appendix C

Artificial Data Generation

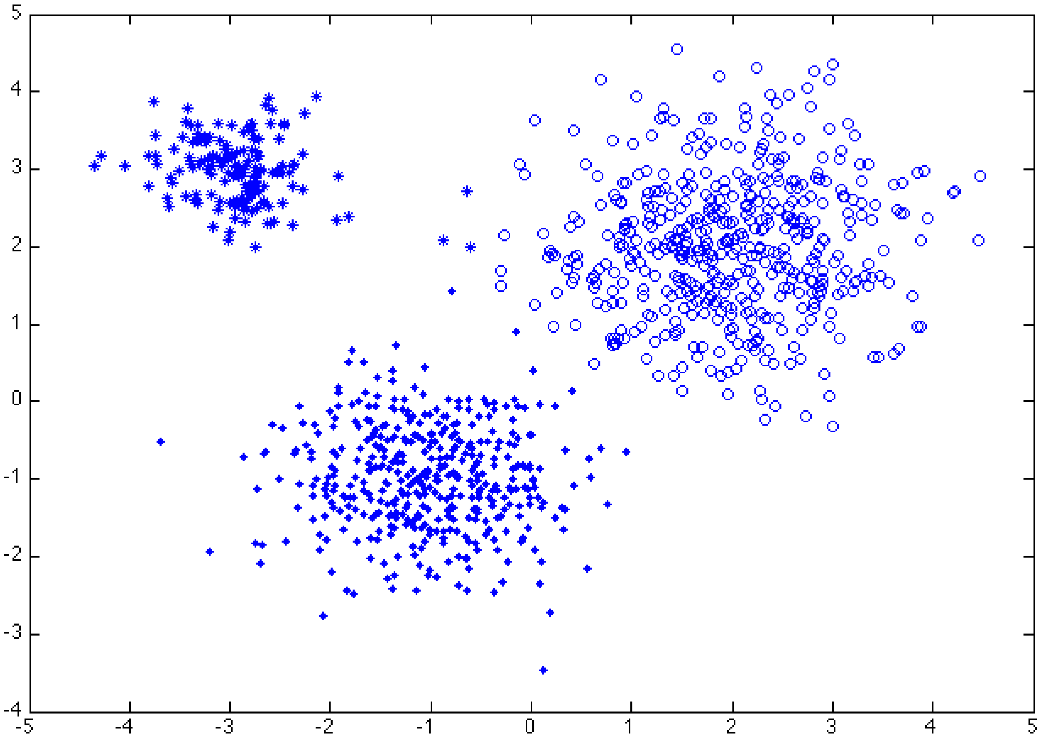
```
% Generate 3 clusters
MU1 = [-1 -1]; SIGMA1 = [.5 0; 0 .5];
MU2 = [2 2];   SIGMA2 = [.7 0; 0 .7];
MU3 = [-3 3];  SIGMA3 = [0.2 0; 0 0.2];

seg1 = mvnrnd(MU1,SIGMA1,2100);
seg2 = mvnrnd(MU2,SIGMA2,2400);
seg3 = mvnrnd(MU3,SIGMA3,800);

X1 = [seg1; seg2; seg3];
```

Appendix D

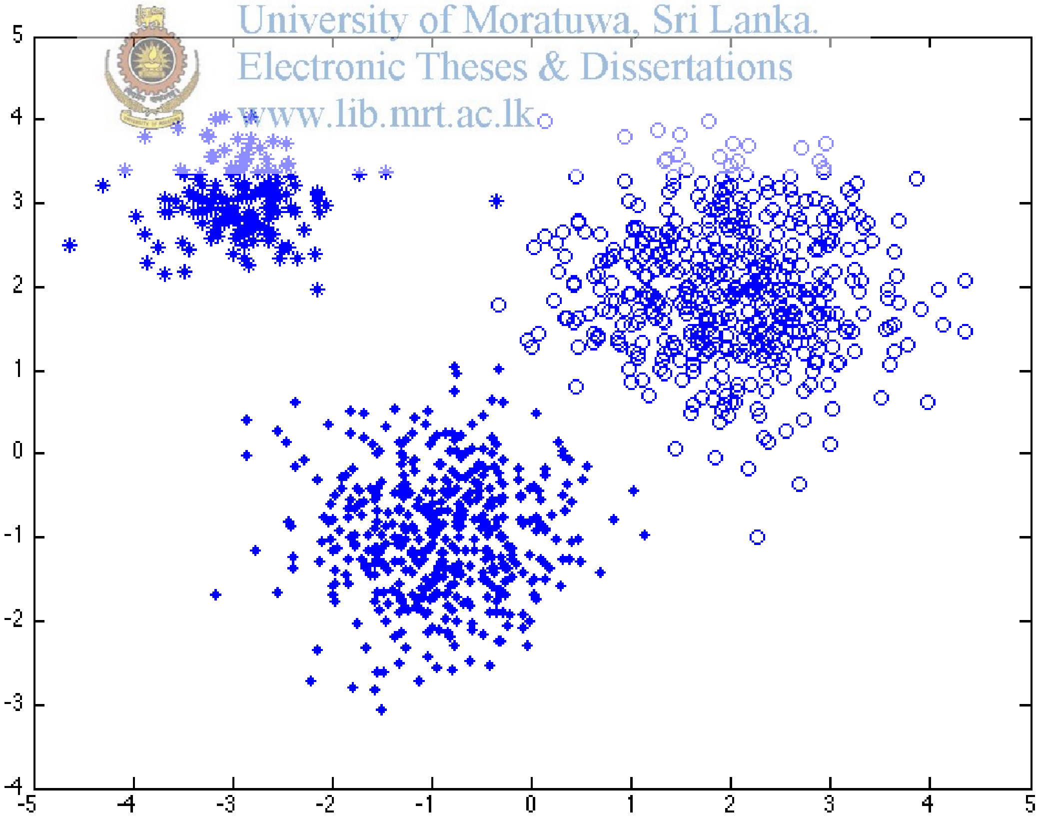
Scatter Diagram of Artificial Data Merging results



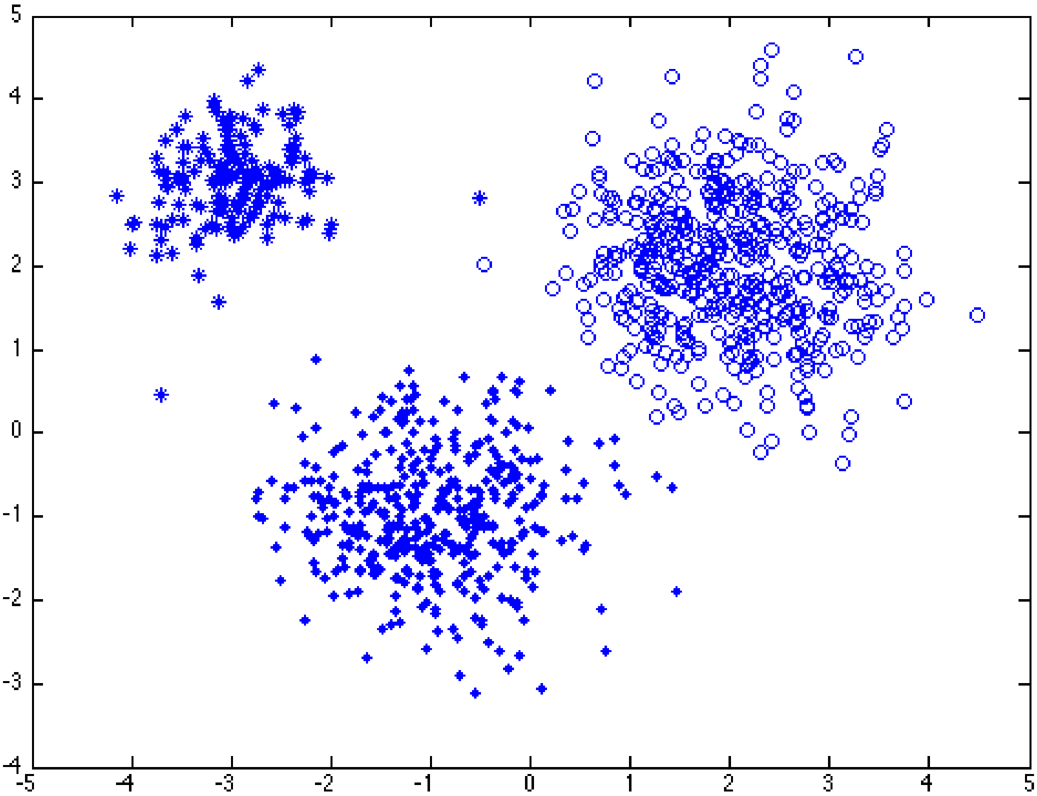
Node 1 Cluster Results



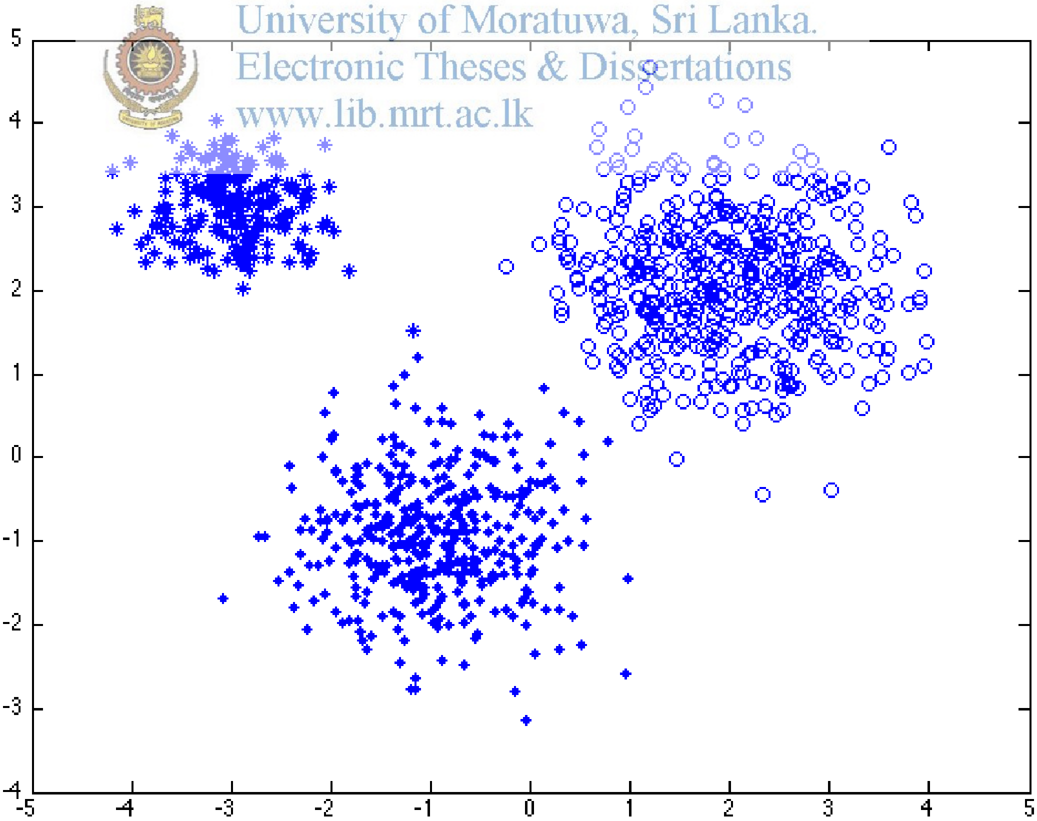
Node 2 Cluster Results



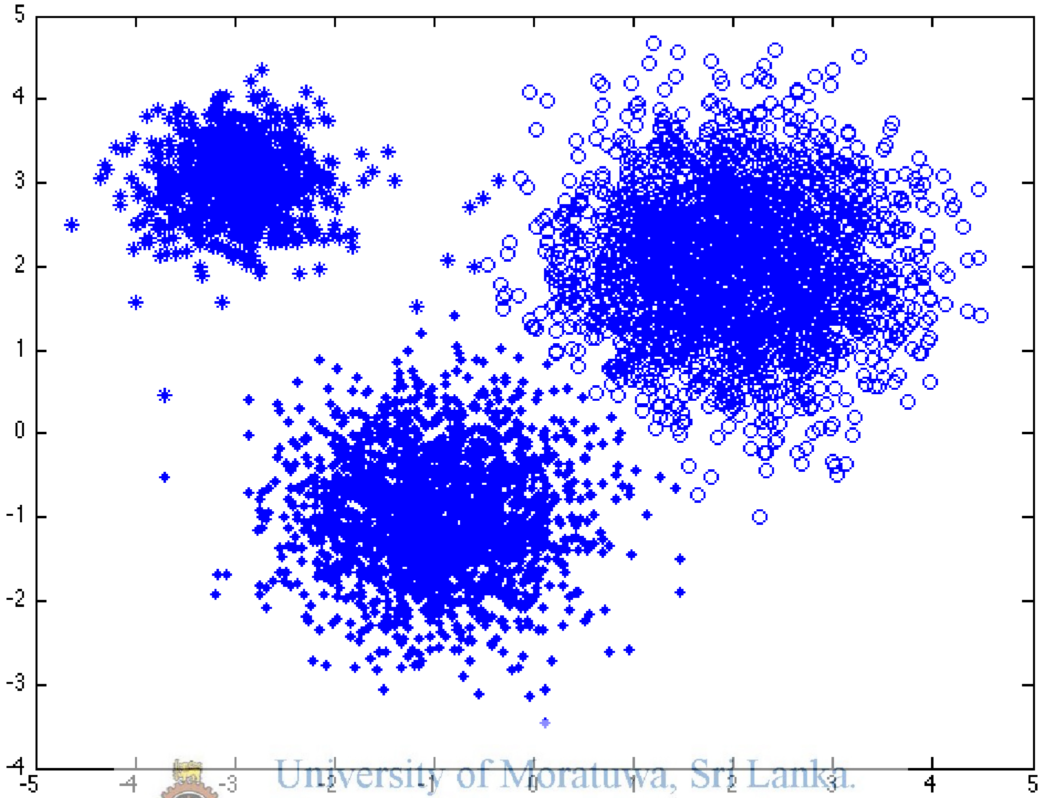
Node 3Cluster Results



Node 4 Cluster Results



Node 5 Cluster Results



Merged result of all 5 nods

Appendix E

Descriptive Statistics of Real-world Data Clustering

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	311.33	251.30	1,152	1
Since Last Purchase	216.40	227.70	1,152	1
Loyalty Point Collected	276.30	262.92	1,360	0
Loyalty Point Redeemed	164.90	153.86	1,320	0

Cluster 1 Descriptive Statistics (N=75,580)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	523.89	241.89	1,152	2
Since Last Purchase	104.21	162.21	1,142	1
Loyalty Point Collected	2,434.40	1,388.60	9,730	810
Loyalty Point Redeemed	241.37	515.72	2,700	0

Cluster 2 Descriptive Statistics (N=7106)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	629.90	253.60	1,152	1
Since Last Purchase	188.50	236.95	1,150	1
Loyalty Point Collected	3,246.95	1,285	10,337	1,274
Loyalty Point Redeemed	2,185.50	1,060.09	5,809	0

Cluster 3 Descriptive Statistics (N=3507)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	791.03	335.75	1,152	50
Since Last Purchase	116.31	181.97	1,136	1
Loyalty Point Collected	18,590.01	6,088.36	50,806	10,489
Loyalty Point Redeemed	164.90	49,162.64	26,553	0

Node 4 Descriptive Statistics (N=356)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	788.88	272.47	1,152	8
Since Last Purchase	137.55	211.21	1,094	1
Loyalty Point Collected	8,292.70	2,520.60	18,514	4,613
Loyalty Point Redeemed	4,797.65	2,774.22	14,009	0

Node 5 Descriptive Statistics (N=1,256)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	539.23	240.30	1,152	1
Since Last Purchase	330.36	274.40	1,152	0
Loyalty Point Collected	736.92	658.66	3,000	0
Loyalty Point Redeemed	160.33	339.92	1,660	0

Node 6 Descriptive Statistics (N=26,212)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	391.51	416.13	1,142	60
Since Last Purchase	74.56	49.26	294	3
Loyalty Point Collected	98,014.28	46,221	234,017	41,313
Loyalty Point Redeemed	69,711	44,092	210,056	1,750

Node 7 Descriptive Statistics (N=39)

Attribute	Mean	Standard Deviation	Max	Min
Since First Purchase	536.78	433.25	1,150	4
Since Last Purchase	80.34	77.81	484	1
Loyalty Point Collected	64,714	70,298	425,057	28,325
Loyalty Point Redeemed	33,151	30,107	184,548	0

Node 8 Descriptive Statistics (N=46)

Appendix F

GPU and GPU Execution Time

# of Data Points	CPU Time In Seconds	GPU Time In Seconds
195	2.24079895	1.874832153
390	4.078645229	2.150387049
781	16.66117287	2.850647211
1562	77.31625414	9.025827885
3125	111.2946901	5.516698837
6250	338.895828	9.107051134
12500	480.0107498	9.128021002
25000	2675.170223	19.49317503
50000	3225.784702	12.18083715
100000	5184.894117	13.60191917